UNIVERSIDAD TÉCNICA PARTICULAR DE LOJA

La Universidad Católica de Loja

ESCUELA DE CIENCIAS DE LA COMPUTACIÓN

"Determinar el nivel de seguridad del Sistema Syllabus desarrollado en el GDS-UTPL, aplicando técnicas de revisión de código y pruebas de penetración que permita determinar vulnerabilidades del sistema"

Tesis previa a la obtención del título de
Ingeniería en Sistemas Informáticos y Computación

DIRECTOR:

Ing. Danilo Rubén Jaramillo Hurtado

CO-DIRECTORA:

Ing. María Belén Mora

AUTOR:

Junior Ulises Sinche Salinas

(2010)

CERTIFICACIÓN

DANILO RUBEN JARAMILLO HURTADO.

DIRECTOR DE TESIS

CERTIFICA:

Haber dirigido y supervisado el desarrollo del presente proyecto de tesis previo a la obtención del título de INGENIERO EN SISTEMAS INFORMÁTICOS Y COMPUTACIÓN, y una vez que este cumple con todas las exigencias y los requisitos legales establecidos por la Universidad Técnica Particular de Loja, autoriza su presentación para los fines legales pertinentes.

Loja, 14 de octubre de 2010

_____

DANILO RUBEN JARAMILLO HURTADO

AUTORÍA

El presente proyecto de tesis con cada una de sus observaciones, análisis, evaluaciones, conclusiones y recomendaciones emitidas, es de absoluta responsabilidad del autor (s).

Además, es necesario indicar que la información de otros autores empleada en el presente trabajo está debidamente especificada en fuentes de referencia y apartados bibliográficos.

.....

F) autor (s)

CESIÓN DE DERECHOS

Yo, Junior Ulises Sinche Salinas declaro conocer y aceptar la disposición del Art. 67 del Estatuto Orgánico de la Universidad Técnica Particular de Loja que en su parte pertinente textualmente dice: “Forman parte del patrimonio de la Universidad la propiedad intelectual de investigaciones, trabajos científicos o técnicos y tesis de grado que se realicen a través, o con el apoyo financiero, académico o institucional (operativo) de la Universidad”.

.....

F) autor (s)

AGRADECIMIENTO

La presente tesis, si bien ha requerido de esfuerzo y mucha dedicación por parte del autor y su director, no hubiese sido posible su finalización sin la cooperación desinteresada de todas y cada una de las personas que me han brindado el apoyo tanto moral como espiritual, muchas de las cuales han sido un fuerte soporte en momentos difíciles.

Primero dar gracias a Dios, por estar siempre presente en cada paso que doy, por fortalecer mi corazón e iluminar mi mente, por haber puesto en mi camino a aquellas personas que han sido mí inspiración durante todo el periodo de estudio.

Agradecer siempre a mi familia, porque se han preocupado de mi bienestar, porque gracias a ellos es que he obtenido un título de tercer nivel. A mis padres Ulises y Rosa, mis hermanos Daniel, Lorena, Mayra, mi sobrino Darío, a mi cuñada Mary, tíos, primos, porque a pesar de todo siempre cuento con su apoyo y alegría que me han brindado durante toda la vida y gracias a ellos puedo seguir adelante.

DEDICATORIA

Esta tesis la dedico con todo amor y cariño a Dios por darme la vida junto a una maravillosa familia.

A mis padres porque me dieron mi carrera, mi futuro, porque siempre creyeron en mi a pesar de la adversidades. Gracias por todo, los amo con toda el alma.

A mis hermanos porque ellos siempre han estado conmigo apoyándome.

A toda mi familia, me gustaría nombrarlos a todos pero simplemente les digo gracias por todo desde mi corazón.

Índice

Introducción

La seguridad del software consiste tanto en la protección de información que se maneja en los sistemas informáticos como el funcionamiento integral del mismo. Uno de los valores agregados más importantes a un sistema informático es la autoprotección, es decir, ¿cómo se va a proteger el sistema y la información que maneja? La protección de información contra cualquier intruso incluye desde la autorización, restricciones de acceso y ejecución de tareas en entornos seguros, de tal forma que todas las actividades relacionadas con las reglas de negocio sean realizadas en su integridad y sin fallos.[1]

El objetivo principal de esta tesis es comprobar qué tan seguro es el SISTEMA DE GESTIÓN ACADÉMICA de la Universidad Técnica Particular de Loja (SYLLABUS-REFACTORY) partiendo del hecho que todo sistema es inseguro, aplicando revisiones de código y pruebas de penetración para determinar las vulnerabilidades. Luego se planteará un acercamiento a una solución factible a todos aquellos problemas que hayan sido descubiertos durante el proceso de ejecución del presente trabajo.

Otro objetivo de la presente investigación es observar cómo se ejecutan los procesos en un ambiente real de desarrollo de software, con el fin de hacer un análisis de cómo se llevan las tareas de desarrollo con respecto a la seguridad.

En la primera parte del capítulo I se trata resumidamente temas generales con respecto a seguridad informática, estos temas no están enfocados directamente en el proceso de desarrollo de software, sino son aspectos externos a este proceso, pero hay que tenerlos en cuenta porque estos problemas pueden afectar al funcionamiento del sistema. En la segunda parte de este capítulo se tomó en cuenta aquellos aspectos que afectan claramente a la seguridad del software y que están directamente relacionadas con el código, que se implementa en la fase de codificación del proceso de desarrollo de software.

En el capítulo II se realizó una revisión de herramientas utilizadas para la ejecución de este proyecto, así como un análisis comparativo para determinar cuáles son las mejores herramientas para la revisión de código y verificar la seguridad del Sistema SYLLABUS-REFACTORY. Las herramientas ayudan a agilizar procesos. Existen aspectos que necesariamente hay que realizarlos de forma manual, porque las herramientas no tienen la heurística necesaria para validar cuáles son los problemas de seguridad.

En el capítulo III se seleccionó aquellos módulos del SYLLABUS-REFACTORY a ser analizados y comprobar su nivel de seguridad, esta selección se basa en el cronograma del sistema. Los módulos seleccionados son los que conforman el núcleo de la aplicación, es decir la parte que permite realizar una matrícula básica. Los módulos del núcleo son:

- El módulo Componentes Educativos, permite la creación y configuración de asignaturas.

- El módulo Programa Académico, permite la creación de carreras.
- El módulo Estructura Curricular, permite la creación de un pensum de estudios que debe cursar un estudiante para obtener un título profesional.
- El módulo Oferta Académica, consiste en ofrecer materias que posteriormente los alumnos podrán tomarlas para completar su curriculum académico.
- Y el módulo Matrícula, es el proceso de inscripción en asignaturas ofertadas por parte de estudiantes, este último proceso involucra la integración con los componentes antes mencionados.

Adicionalmente en el capítulo III se aplicaron los procesos para comprobar que tan seguro es el sistema SYLLABUS-REFACTORY y se ejecutaron procesos para las revisiones de código, inyecciones SQL, inyecciones XSS, otros y se verificó el correcto manejo de excepciones. La comprobación se baso en los siguientes puntos:

- Los bugs de seguridad están enfocados en el correcto manejo de excepciones para evitar fugas de información.
- Comprobar que todos los datos de entrada contengan validaciones, con el fin de evitar inyecciones de diversos tipos como XSS, SQL, otros para evitar la manipulación incorrecta de datos, negación de servicios o cualquier otro problema de seguridad.
- Correcta ejecución de sentencias o procedimientos almacenados con el fin de evitar inyecciones SQL o la manipulación inadecuada de datos.
- Para las inyecciones XSS se hicieron pruebas con la plataforma de la interfaz gráfica de usuario (EXTNET), haciendo inyecciones de scripts y ejecutándolos en la interfaz gráfica de usuario.

El capítulo IV se enfoca en los problemas de seguridad encontrados en el capítulo IV, cómo contrarrestar sus efectos, de tal forma que se tenga solución a problemas de seguridad que pueden afectar al SYLLABUS-REFACTORY. Cada problema de seguridad en cada una de las capas del sistema tiene solución y será estandarizada para que en otros sistemas se aplique la misma solución. El estándar es para comprobar ¿qué tan seguro es un sistema partiendo del hecho que todo sistema es inseguro?

En las conclusiones y recomendaciones se encuentran los principales puntos a los que se ha llegado en este estudio, en base a los diversos procesos que se ejecutaron durante la realización de la presente tesis.

Al final se han agregado los anexos de documentos y archivos que se ha utilizado para poder realizar cada una de las tareas y validaciones durante el proceso de ejecución de esta tesis. Los archivos contienen toda la información que se recopilo del sistema SYLLABUS-REFACTORY.

Nota: Algunas imágenes que se pueden visualizar en el presente trabajo han sido cortesía del proyecto SYLLABUS-REFACTORY.

1. Capítulo I. Revisiones generales sobre seguridad.

En el presente capítulo se realiza una introducción a los problemas tanto de seguridad informática como a los de seguridad del software. La seguridad informática es tratada de forma resumida. Los problemas de seguridad del software son en los que se enfoca el presente proyecto, es decir las vulnerabilidades del software que se pueden encontrar en el código fuente del sistema.

"La Seguridad Informática se define, como la estructura de control establecida para gestionar la disponibilidad, integridad, confidencialidad y consistencia de los datos, sistemas de información y recursos informáticos. Conjunto de controles que tienen la finalidad de mantener la confidencialidad, íntegridad y confiabilidad de la información."[1]

El objetivo de la seguridad informática es minimizar la probabilidad de ejecutar cualquier acción que comprometa el funcionamiento normal de los procesos de una empresa y la integridad de la información que se encuentra en un sistema informático, en otras palabras es eliminar toda amenaza posible para un sistema de información.

En la primera parte de este capítulo se tratan temas generales que influyen en la seguridad informática, a estos temas se da una revisión rápida. En la segunda parte se tratan temas relacionados con la seguridad del software, estos son problemas que pueden ser corregidos netamente en el código fuente de un sistema.[2]

1.1. Problemas de seguridad informática.

Al hablar de seguridad informática hay que tener en cuenta todo lo que puede comprometer al correcto funcionamiento del sistema de información, existen problemas que están involucrados con: aspectos físicos, manejo de datos lógicos y el personal.[2]

Seguridad física.

La seguridad física consiste en la "aplicación de barreras físicas y procedimientos de control, como medidas de prevención y contramedidas ante amenazas a los recursos e información confidencial. Se refiere a los controles y mecanismos de seguridad dentro y alrededor del Centro de Cómputo así como los medios de acceso remoto al y desde el mismo; implementados para proteger el hardware y medios de almacenamiento de datos."[2]

La seguridad física hace referencia a toda la infraestructura, por ejemplo ubicación física, equipos de trabajo, instalaciones eléctricas, etc.. En ocasiones la infraestructura no es la más adecuada, debido a que existe humedad o polvo, no tienen una instalación a tierra,

[1]Tomado de: http://www.forosdelweb.com/f20/que-seguridad-127115/
[2]Seguridad física. Tomado de: http://www.segu-info.com.ar/fisica/seguridadfisica.htm

hacen falta reguladores de voltaje o ups, etc., los cuales son factores que influyen en el equipamiento haciendo que estos se estropeen, lo que conlleva a perdidas de información o de activos de la empresa. Muchas de veces es difícil recuperarse de pérdidas de datos lógicos o activos físicos.[3]

Seguridad en servidores.

Los servidores son equipos con gran capacidad de procesamiento que permiten el acceso a recursos compartidos como información o hardware. La ubicación de una sala de servidores debe ser en un lugar donde únicamente tengan acceso físico las personas encargadas, la ubicación de la sala debe ser estratégica con el fin que se pueda monitorear la entrada y la salida de las personas autorizadas para estar en ella.[4]

Los respaldos de información deben estar en lugares adecuados, se recomienda tener más de una copia de seguridad y cada una de estas debe estar en un lugar distinto, es decir físicamente alejado. [2]

Puertos: Los puertos son protocolos que permiten el acceso a servicios en los servidores. En un servidor se debe tener abiertos los puertos estrictamente necesarios, para brindar el servicio para el que ha sido encomendado dicho equipo, no solo por optimización de recursos (memoria, procesador, etc.) sino también por seguridad, porque cada puerto abierto es un punto de acceso adicional a los diferentes servicios.[5]

Una forma de evitar tráfico por puertos no deseados es a través de listas de control de acceso, permitiendo o denegando tráfico de red. Las listas de control de acceso deben ser creadas de acuerdo a las necesidades de la empresa y deben controlar el tráfico entrante y saliente. También existen herramientas gratuitas las cuales son muy fáciles de utilizar, como el NMAP, el mismo que permite escanear un equipo y determinar qué puertos están abiertos.[2][6]

Ataques de DoS (Negación de servicios): Los ataques de negación de servicios consisten en hacer muchas solicitudes a un servidor, por lo tanto el servicio llega colapsar y cuando un usuario real del sistema quiere hacer uso del servicio la petición es puesta en espera por el servidor, debido a que el servidor está atendiendo las peticiones no validas. En ocasiones las peticiones del usuario real son rechazadas porque el servidor no avanza a cubrir todas las solicitudes.[2]

Puertas traseras: Las puertas traseras son puntos de acceso a un sistema donde no se detecta la intrusión, debido que el creador de la puerta pudo haber dado privilegios para el acceso o se está entrando al sistema como administrador. Estas muchas veces son utilizadas con fines maliciosos, como el plagio de información y espionaje. Las puertas traseras son creadas con fines de administración de un sistema por parte del desarrollador

y se vuelve un problema cuando el desarrollador o alguien que conozca de la puerta la explotan en su beneficio.[2]

Problemas de seguridad con software malicioso.

De los programas maliciosos los más conocidos son los virus, los cuales son software que produce un mal funcionamiento en los sistemas, ocasionando lentitud en la ejecución de procesos, errores en el sistema operativo, abren o cierran puertos, anulan servicios, ocupan memoria, ejecutan acciones no deseadas o permiten el plagio de información.

Virus: Un virus tiene tres funciones principales las cuales que son: ocasionar un daño, autoprotección y reproducción o expansión. Si un programa no tiene una de las operaciones antes mencionadas no es catalogado como un virus. La mejor solución para eliminar virus es tener un antivirus actualizado.[2]

Espías: Los espías son software que permiten el plagio de información de un computador. Los espías están monitoreando todo movimiento que se hace en un computador y dichos movimientos los almacenan en un archivo localmente, por lo general guardan toda información ingresada por teclado, esta información es nombres de usuarios, claves, otros textos. Los espías capturan toda información ingresada y luego es enviada a la persona que esta espiando.[2]

Seguridad en el recurso humano.

Personal: El personal es el recurso más valioso que posee una empresa, pero también es el aspecto más influyente en la seguridad, el personal son quienes manejan los procesos de una empresa, por ende de ellos depende la supervivencia de la misma.

Muchas veces el acceso de personal no autorizado a ciertas áreas, usuarios no capacitados, usuarios mal intencionados, personal despedido, etc. pueden convertirse en problemas de seguridad que debe afrontar una empresa, cada uno de ellos representa un peligro potencial, debido al conocimiento que poseen con respecto a los procesos de la misma.[7]

El personal con la debida capacitación y comprometido con la empresa, minimiza el riesgo en muchos ámbitos, porque ellos ayudan a cuidar los activos. El personal que no está comprometido se lo tiene que comprometer de alguna forma, por ejemplo con incentivos (económicos, capacitaciones), reuniones de socialización, con un ambiente de trabajo agradable.[8]

Personal externo: Se debe confiar en el personal que labora en la empresa, pero que hay de las personas que ajenas a esta, ¿cuáles son los fines de su llegada o con que intenciones ingresan? Siempre hay que tener cuidado en las acciones que realizan personas particulares, porque pueden de alguna forma ingresar a áreas restringidas y pueden tener acceso a información propia de la empresa.[8]

Personal informático: Son las personas encargadas del manejo de la información lógica. Ellos administran los sistemas y ponen las reglas para el manejo de seguridad en la información, también conocen los procesos de la empresa. Son personas capacitadas en temas informáticos y conocedores del funcionamiento de procesos de negocio de la empresa.[8]

Los ataques que se realizan a los sistemas, son realizados por personas que conocen la empresa, es decir pueden ser trabajadores de esta o son personas que por alguna razón dejo de trabajar en dicha empresa, que por cualquier situación cometen actos que van en contra de la integridad de la empresa. Como medida de seguridad cuando un empleado termina su relación laboral con la empresa se recomienda bloquear sus cuentas de usuario para el acceso a los sistemas de información.[7]

1.2. Diferencia entre seguridad del software y seguridad informática.

La seguridad informática implica tener en cuenta aspectos físicos, equipamiento tecnológico, sistemas operativos, programas antivirus, el personal, con el fin de tener un sistema de información seguro, todos los aspectos que influyen en dicho sistema deben trabajar sincronizadamente para evitar problemas que a futuro conllevan al funcionamiento incorrecto del sistema y por ende a que la empresa como tal no cumpla sus objetivos de seguridad de forma adecuada.

En cambio, la seguridad del software está enfocada al ciclo de vida de un sistema, a tener desde la fase de análisis hasta la fase de implementación políticas claras para la estandarización de procesos. Mientras se desarrolla un sistema informático hay que tener en cuenta problemas que pueden poner en riesgo la ejecución de las fases del proceso de desarrollo de software para darle solución oportuna.

1.3. Problemas de seguridad inmersos en el desarrollo de software.

A pesar de haber superado los problemas de overflows en lenguajes de alto nivel se producen nuevos problemas de seguridad con el software. Conforme avanza la tecnología a si mismo avanzan las técnicas para violar la seguridad de los sistema.

Muchos problemas de seguridad se encuentran en el código fuente de un sistema. Existen problemas como excepciones no controladas, inyección SQL, inyecciones de scripts(Cross-site scripting), falta de control en procesos de autenticación y autorización, contraseñas fáciles de descifrar. A estas vulnerabilidades se las puede prevenir a través de la práctica de buenas técnicas de programación e implementación de estándares al momento de escribir el código del sistema.

Para tener una aplicación segura hay que tener en cuenta las validaciones de todos los datos de entrada, el control de excepciones, revisiones que código, implementación de algoritmos para encriptación de datos, buen manejo de los procesos de autorización y atentación, manejo de contraseñas, etc. .

1.3.1. Problemas de seguridad de software, Overflows.

Uno de los problemas más comunes de seguridad en lenguajes como C o C++ han sido los overflows (desbordamientos de memoria) que consiste en sobrepasar la capacidad de cálculo o exceso de datos de una variable, esto es un error en la programación. Cuando el desbordamiento es producido intencionalmente por alguien que envía datos que incluyen porciones de código que son ejecutadas posteriormente se convierte en un problema de seguridad.[9]

Una solución para evitar overflows es la utilización de lenguajes de alto nivel (C Sharp, Java, etc.) en cuyas plataformas ya tiene controlado estas situaciones, el programador no debe preocuparse por un overflow, porque el control de la ejecución del programa lo tiene la plataforma del lenguaje en el caso se java su máquina virtual o en .NET el Framework. Las plataformas de los leguajes de alto nivel abstraen el acceso directo a memoria, a diferencia de C o C++ donde es el programador el que tiene mayor control de las sentencias que escribe y es el mismo desarrollador quién debe encargarse del control y manejo de memoria.[9]

1.3.2. Excepciones no controladas.

"Una excepción es un evento que ocurre durante la ejecución del programa que interrumpe el flujo normal de las sentencias."[3]

Una excepción se da cuando se ejecuta un proceso y de forma inesperada ocurre un error debido a una situación que no ha sido prevista o no está debidamente controlada de allí el término "las excepciones son condiciones excepcionales que pueden ocurrir dentro del programa durante su ejecución"[4].

El no capturar una excepción es una problema de seguridad del sistema, porque esta al ser lanzada envía información del sistema, la cual contiene el motivo de la excepción con una descripción detallada de nombres de variables y sus respectivos valores, sí se trata de una excepción de base de datos se muestran nombres de tablas o procedimientos almacenados de donde se produjo la excepción.[10]

[3]Obtenido de: http://programacion.com/java/tutorial/excepciones/
[4]Obtenido de: http://www.zator.com/Cpp/E1_6.htm

Sí una excepción no está controlada esta presenta el mensaje de error a través de la interfaz gráfica de usuario. En java la máquina virtual tiene un manejador de excepciones incluido el cual se encarga de capturar la excepción aunque el programador no haya escrito el bloque para capturar excepciones, por tal razón en java no se presentan las excepciones a través de la interfaz gráfica de usuario, lo que no ocurre en lenguajes como C Sharp, C o C++, en estos lenguajes es necesario especificar los bloques para capturar una excepción cuando esta es lanzada.[11]

Se puede guardar la excepción en un log o una base de datos con el fin de que sea revisada posteriormente por el administrador del sistema para que haga los correctivos necesarios y mantener un historial de excepciones ocurridas.

Cuando un usuario común está utilizando una aplicación y ocurre una excepción la cual no ha sido controlada, el usuario por lo general no a saber qué hacer, su primera sensación es que el sistema está con problemas o tal vez al usuario le cause una fuerte impresión por el error dejándolo en shock por un momento. Sí a la excepción se la trata de forma adecuada y presentando un mensaje personalizado el usuario va a saber qué debe hacer. La captura de excepciones se debe hacer por motivos de seguridad y la personalización de mensajes es por usabilidad.[12]

La mejor forma de controlar las excepciones es tener siempre los bloques try-catch-finalize los cuales permiten capturar los errores y tratarlos de forma adecuada, al usuario hay que presentarle un mensaje personalizado y no el error como tal. Cuando se captura una excepción es recomendable presentar un mensaje dependiendo del tipo de excepción o también se puede presentar un mensaje estándar para todas las excepciones.[13]

La captura de excepciones da mayor control al sistema, permitiendo a este la tolerancia a fallos, de tal forma que pueda recuperarse de un error, disminuyendo las vulnerabilidades y dando como resultado un sistema con un mayor grado de confiabilidad.[13]

1.3.3. Inyecciones

Inyecciones SQL: Son un problema común en sistemas que utilizan base de datos, consisten en modificar las sentencias SQL que son ejecutados en una base de datos. Por lo general la modificación de sentencias se lo hace a través de la interfaz gráfica de usuario, urls, cookies, etc.. Una vez producida la inyección existe inconsistencia de datos o la falla del sistema. Las inyecciones no son hechas por usuarios comunes sino por personas que ya tienen la intención de penetrar al sistema.[14] [15]

Por ejemplo: Se tiene dos campos para ingresar datos: usuario y clave, donde se ingresa:

Usuario : ‘ or ’1’=’1
Clave: ‘ or ’1’=’1

Además existe la siguiente sentencia

Select usuario from usuarios
where Usuario = 'parámetro_usuario' and clave='parámetro_clave'

La sentencia concatenada con los datos ingresados queda la siguiente manera

Select usuario from usuarios where
usuario = '' or '1'='1' and clave='' or '1'='1'

Se observa que 1 es igual a 1 por lo tanto se tiene un usuario valido, razón por la cual se puede ingresar al sistema sin ningún problema.[16]

Una solución para evitar las inyecciones SQL es la utilización de procedimientos almacenados, porque al llegar a la base de datos las cadenas de caracteres se ejecutan tal cual fueron enviadas, pero hay que tratar de evitar los procedimientos almacenados que contienen sentencias SQL dinámicas, porque estos se forman a través de la concatenación de cadenas de texto en la base de datos.[15]

1.3.4. Cross-site scripting (XSS).

"Su nombre original es Cross Site Scripting y es abreviado como XSS para no ser confundido con las siglas CSS, (hojas de estilo en cascada). Las vulnerabilidades de XSS originalmente abarcaban cualquier ataque que permitiera ejecutar código de scripting, como VBScript o Java Script, en el contexto de otro sitio Web".[5]

Las inyecciones de scripts es una nueva forma de ataque en ambiente Web, el cual consiste en el ingreso de etiquetas HTML y de código en javascript o VBScript para que se ejecuten en el navegador Web impidiendo la ejecución normal de la aplicación, llevando incluso a la denegación de servicios.[17]

Para el siguiente ejemplo se tiene una página Web donde se va a guardar un registro, a través del campo de descripción se ingresa el siguiente texto:

<script>alert ("Hola Mundo")</script>

Cuando posteriormente se desee acceder a la información del campo a través del navegador lo que va a presentarse es un mensaje que dice "Hola Mundo".

Van a existir sentencias con mayor complejidad que serán ingresadas, por ende van a ejecutar procesos complejos, lo que puede terminar en denegación de servicios, por lo tanto no se podrá hacer la utilización del sistema. Al texto anterior se le agregó un ciclo repetitivo quedando así:

<script>for(var i=0; 1==1; i++){alert ("Hola Mundo")}</script>

[5]Obtenido en: http://dspace.espoch.edu.ec/bitstream/123456789/325/1/18T00406.pdf

Esta información al ser recuperada en un formulario (no es necesario que sea el mismo) será ejecutado el script, dando como resultado la aparición de infinitas veces el mensaje de alerta: "Hola mundo", se ve en el ciclo for una condición que siempre va a ser verdadera, así que jamás dejará de aparecer el mensaje.[18]

La mejor forma de evitar los ataques de inyecciones en general es siempre validar toda la información que se ingresa a través de controles que se encuentran en la interfaz gráfica usuario, esto se debe hacer tanto en el cliente como en el servidor.[19]

1.3.5. Autenticación.

La autenticación permite la identificación de usuarios de un sistema, es una forma de saber quién ingresa y qué rol tiene la persona que está ingresando (Administrador, secretario, contador, gerente, etc.).

La autenticación generalmente se lo hace a través de un usuario y contraseña, el usuario puede ser de dominio público, pero la contraseña es de carácter privado. Por ejemplo un id de usuario es el identificador de su cuenta de correo el cual muchas personas lo conocen, pero la contraseña del correo de una persona en particular es conocida solo por la persona dueña de esa cuenta.[20]

Contraseñas inseguras, fáciles de descifrar: Las contraseñas son usadas conjuntamente con un login para la autenticación de usuarios. La autenticación es la forma de identificación de usuarios, es para saber si un usuario es quien dice ser, su login es único y la contraseña es la palabra clave que sirve para verificar su veracidad, es decir es su pasaporte al sistema.

Las contraseñas inseguras son fáciles de descifrar y facilitan el acceso a intrusos. Se recomienda no tener una contraseña igual al usuario (usuario = usuario1, clave = usuario1), tampoco tener de contraseña una secuencia de caracteres (123, abc, 456, etc.) o algún dato personal (Nombre, fecha de nacimiento, nombre de personas allegadas, etc.) ni combinaciones entre estos datos.

Sí se desea tener contraseñas seguras y difíciles de descifrar para un atacante, se debe tener en cuenta una cantidad mínima de caracteres, con una combinación de números, letras y otros símbolos (#, $, %, &, /, @, ¿, etc.), aplicando estos factores depende que tan segura es una contraseña.

Es recomendable tener distintas contraseñas por cada cuenta que tenga un determinado usuario y renovarla con cierta periodicidad de tal forma que sí esta es descubierta ya no será útil, así se impide el acceso a las distintas cuentas que posea el usuario.

Se debe crear políticas para la creación y renovación de contraseñas, donde intervenga la vigencia de una contraseña y la complejidad de las mismas, dando un mayor grado de

seguridad con el fin de evitar accesos no autorizados. Algunas de las políticas que se deben tener son: que cada usuario tenga su propio login y contraseña, no se deben compartir contraseñas entre usuarios, estas son privadas. Tampoco se debe tener un login para más de un usuario.

Contraseñas en base de datos almacenadas sin encriptar: Las contraseñas almacenadas en tablas de una base de datos deben estar encriptadas, para que las personas que tenga acceso a estas tablas no puedan verlas. Aplicar algoritmos de encriptación (hash, md5, etc.) es una forma recomendable para almacenar las contraseñas, al estar encriptada la información es difícil de leer y entender, a pesar que existe la forma de forzar estas contraseñas, estos procesos pueden tardar días o meses dependiendo de la complejidad de la contraseña y de cómo se la haya encriptado.[21]

1.3.6. Autorización.

La autorización es el mecanismo que sirve para verificar sí se tiene el permiso para ingreso a un determinado sitio y que se permite hacer una vez autenticado.

La autorización va de la mano con la autenticación porque un usuario que ya se ha identificado se puede saber sí está autorizado realizar ciertas acciones (o que se le permite hacer) y a donde se le permite acceder, los privilegios que posee, puede o no puede ver cierta información, se le permite modificar o no ciertos registros, etc..

Observación.

Conforme avanza la tecnología surgen nuevos problemas de seguridad, cada problema nuevo debe ser analizado y corregido de la mejor manera, con de fin de evitar que se convierta en un error que afecten gravemente al correcto funcionamiento de un sistema de información. Los temas que se presentan en este trabajo son vulnerabilidades que se explotan actualmente en muchos sistemas de información.

Cada vulnerabilidad es un error que afecta a la parte operativa de una empresa, porque retrasa las operaciones que se realizan con los sistemas de información. Al crear un producto libre de fallas, da prestigio a la empresa desarrolladora y crea confianza en los usuarios finales, porque un producto de alta calidad tiene mejores prestaciones.

Muchas de las vulnerabilidades que se mencionan en el presente capítulo no son tomadas en cuenta al momento de desarrollar software. Se realizan pruebas a los sistemas, pero no se revisan estas vulnerabilidades.

2. Capítulo II. Análisis de herramientas para detectar vulnerabilidades en el software.

En el presente capítulo se realiza una revisión y análisis de las herramientas que se utilizan para la revisión de código en las diversas capas de una aplicación y detectar los posibles bugs de seguridad que pueden existir en un sistema de información.

La revisión de código es la técnica con mayor efectividad para encontrar las vulnerabilidades en el software, pero esta técnica toma demasiado tiempo al menos realizándola manualmente, por tal motivo es necesario apoyarse en herramientas automática.

Entre las temas a verificar con la revisión de código están: las validaciones de todos los datos de entrada; verificar el control y manejo de excepciones para impedir las fugas de información, revisar la forma de ejecución de sentencias en las bases de datos.

2.1. Revisiones de código.

La revisión de código es la técnica con mayor efectividad para detectar vulnerabilidades en una aplicación, esta técnica ayuda a optimizar el código y a tener un producto de alta calidad y seguro. Existen herramientas automáticas que ayudan al proceso de revisión de código las cuales se detallan a continuación.

2.1.1. Resharper.

Es la herramienta más utilizada y popular en el ambiente .NET, esta es una herramienta poderosa la cual permite la revisión de código, refactorización, da sugerencias para optimización de código, bookmarks, etc..

Resharper es compatible con las tecnologías de .NET, como el Visual Studio 2005 en adelante y lenguajes como C Sharp o Visual Basic, también tiene soporte para ASP y ASP MVC.[22]

Esta herramienta es útil tanto para el desarrollo de software como para revisiones de código. Resharper posee intellisence avanzado el cual no solo sugiere la creación de nuevas clases y métodos, sino también sugiere el posible código optimizado. Resharper permite interactuar con las diversas tecnologías de Microsoft.[22]

2.1.2. CodeRush.

Esta herramienta es popular en el ambiente .NET, debido a su gran poder. Es una herramienta que permite un fácil control y refactorización de código, es una de las herramientas más utilizada en el mundo de .NET.[23]

Coderush es similar a Resharper porque poseen funcionalidades similares y da un performance similar en la mayoría de sus utilidades. Coderush posee una versión exprés

(gratuita) con las mismas funcionalidades básicas e igual de eficiente que la versión pagada, esta herramienta es ideal para desarrolladores independientes o empresas pequeñas de software.[23]

2.1.3. FxCop y StyleCop.

FxCop es una herramienta libre creada por Microsoft. Esta herramienta es de análisis estático, es decir permite analizar archivos compilados, por ejemplo archivos .exe o .dll.[24]

StyleCop también es una herramienta libre creada por Microsoft, esta herramienta realiza un análisis dinámico, es decir permite analizar archivos de código fuente, ejemplo archivos cs o vb. Esta herramienta complementa la labor del FxCop y viceversa.[25]

2.1.4. Comparación de herramientas para revisiones de código.

Tabla 1: *Tabla comparativa entre herramientas para la revisión de código.*

	Resharper	CodeRush	fxCop	StyleCop
Compatibilidad con Visual Studio	Si	Si	Si	Si
Propietaria o gratuita	Pagada ($342.02)	Pagada ($249.99) (versión exprés gratuita)	Gratuita	Gratuita
Refactoriza	Si	Si	No	No
Rendimiento	Consume gran cantidad de recursos	Consumo moderado de recursos	Consume pocos recursos	Consume pocos recursos
Soporte ASP.NET	Si	Si	No	No
Tipo de análisis de código	Dinámico	Dinámico	Dinámico	Estático
Analiza el código mientras se lo escribe	Si	No	No	No
Acceso rápido a referencias	Si	Si	Si	No
Optimiza el código	Si	No	No	No
Muestra errores sin necesidad de compilar	Si	No	No	No
Indentación de código	Si	Si(completa con líneas de colores)	No	No

De la comparación realizada entre las herramientas para la revisión de código, se ha llegado a la conclusión que Resharper es la herramienta con mayores prestaciones, a pesar de sus desventajas, esta herramienta ayuda a la revisión de código debido a la eficiencia y fácil acceso a las respectivas clases para poder ejecutar las distintas validaciones. Todas las

herramientas analizadas pueden trabajar conjuntamente en un proyecto de Visual Studio sin presentar problemas de incompatibilidad, porque todas son plugins propios para Visual Studio. Coderush con la indentación de código tiene algo adicional, este plugin tiene líneas de colores que ayuda a la visualización del código, en grandes regiones de código es útil para un mejor entendimiento del código. La comparación entre las distintas herramientas es apreciada en la Tabla 1.

Después de haber realizado un proceso exhaustivo de revisión de herramientas para la revisión de código, se utilizará Resharper y CodeRush, porque lo que se analiza es código dinámico, es decir los archivos fuente del código. Estas herramientas serán utilizadas para la revisión tanto de lógica de negocio como de interfaz gráfica de usuario.

2.2. Herramientas para revisar procedimientos almacenados.

2.2.1. TOAD version 9.7

TOAD es una aplicación para la administración de bases de datos, la cual tiene una licencia propietaria. Con esta herramienta se puede crear y ejecutar procedimientos almacenados.[26]

Ventajas

- Permite la ejecución de paquetes almacenados.
- Genera diagramas de la base de datos.
- Permite la generación de consultas.
- Se puede ejecutar un debug de los procedimientos almacenados.
- Muestra un mapa de los paquetes con cada uno de sus procedimientos almacenados.
- Soporte para Windows.

Desventajas

- Licencia propietaria. El costo de la licencia es de $ 799.

2.2.2. SQL Inyections.

Esta herramienta fue creada con el propósito de realizar pruebas de inyección SQL.

Ventajas

- Licencia Creative Commons, es totalmente gratuita para su uso.
- Creada en .NET lo que le da compatibilidad con ASP.NET.
- Es compatible con distintos lenguajes de manipulación de datos.
- Está orientada para páginas Web. Fácil de utilizar.

Desventajas

- No se la puede utilizar en aplicaciones de escritorio.

Para la revisión de paquetes de base de datos se utiliza el TOAD, debido a que es una potente herramienta para la administración de bases de datos ORACLE y el SYLLABUS-REFACTORY posee este tipo de base de datos. Las ventajas que ofrece TOAD sobre las otras herramientas es muy amplia.

2.3. Revisión manual de código.

Una de las cosas que no se debe olvidar es que con la automatización de procesos de desarrollo de software solo se agiliza algunas tareas, puesto que es complejo encontrar vulnerabilidades en productos de software que tienen grandes cantidades de código, además las herramientas de revisión automática no tienen la heurística suficiente para la toma de decisiones, en cuanto detectar vulnerabilidades de seguridad.[27]

Ventajas.

- Existe mayor precisión en la revisión.
- Un experto en revisiones de código se acopla fácilmente a cualquier lenguaje.
- Las sugerencias para las correcciones son en base a la experiencia de la persona que revisa las vulnerabilidades.
- Se decide que revisar, para dar prioridad a determinados procesos.
- Los resultados son confiables.
- Se puede apoyar de cualquier herramienta para agilizar el trabajo.

Desventajas.

- El tiempo, al menos para revisar en grandes cantidades de código.
- Tiene mayor costo porque es un recurso humano más.

Observación.

Una de las cosas que se debe hacer primero es conocer que herramientas existen con el fin de realizar las revisiones de código, luego conseguir los instaladores y probarlas. La mayoría de las herramientas que tienen un valor monetario, tienen versiones de prueba por un determinado tiempo, también existen las versiones exprés, lo que facilita la obtención de estas herramientas para probar como pueden satisfacer las necesidades de cada usuario.

Luego del análisis y tomando en cuenta que la mayoría de las herramientas son gratuitas y la compatibilidad entre las mismas, es posible utilizar todas estas herramientas en un mismo proyecto, porque cada una ofrece ciertas ventajas que otras herramientas no ofrecen, de esta forma se complementa una a otra.

Cada herramienta tiene alguna ventaja sobre otra, usándolas en conjunto se optimiza el tiempo de revisión de código. Para revisar código en base a la seguridad del sistema se lo debe hacer manualmente, porque ninguna herramienta detecta este tipo de errores. Muchos errores se los detecta ejecutando los procesos, otros se los detecta revisando el código, pero no importa la forma que se detecte los errores lo importante es eliminarlos.

Existen herramientas que pueden ser utilizadas en sistemas escritos en lenguajes distintos a C#, pero el objetivo de la revisión siempre será encontrar vulnerabilidades en el código.

3. Capítulo III. Selección de módulos y revisiones de código.

En el presente capítulo se realiza la selección y revisión de código de los módulos del SYLLABUS-REFACTORY. Se ha revisado las diversas capas de la aplicación y se han encontrado diversos bugs de seguridad, los cuales se detallan en los anexos del presente trabajo. En base a los bugs encontrados se determina el que tan seguro es el sistema de SYLLABUS-REFACTORY.

3.1. Selección de módulos para comprobar su seguridad.

La selección de los módulos se enfoca en la primera liberación del sistema SYLLABUS-REFACTORY, la cual corresponde a la funcionalidad básica de matrícula de estudiantes nuevos.

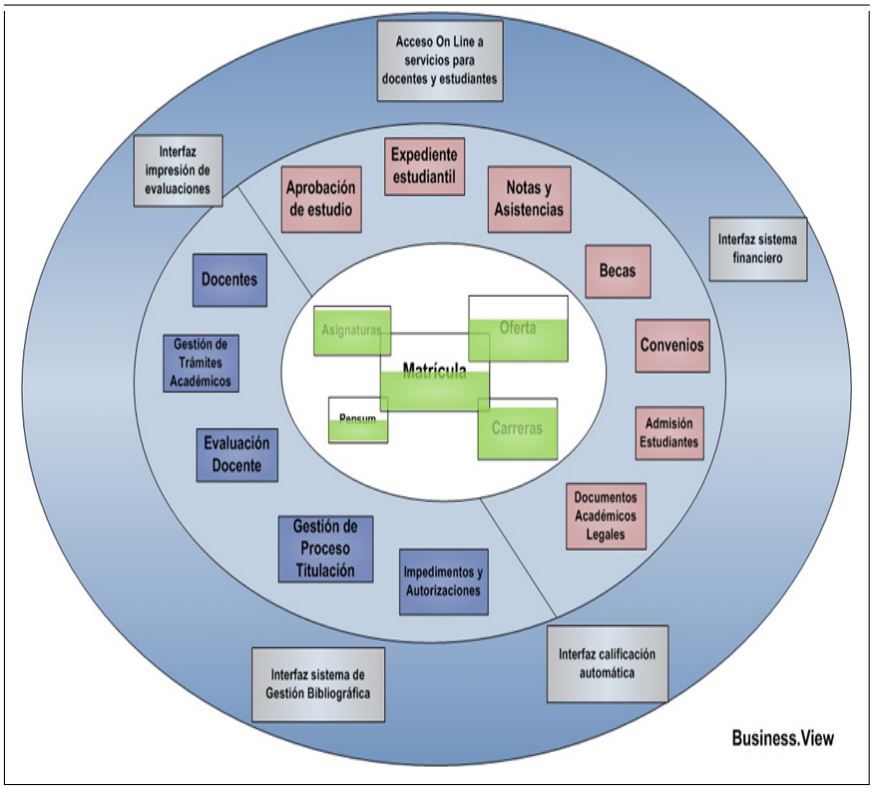


Figura 1: *Diagrama general de los módulos que componen al sistema SYLLABUS-REFACTORY.*

En la Figura 1 se observa un diagrama general de los módulos que conforman el SYLLABUS-REFACTORY[6], la parte central de esta figura es el núcleo del sistema. El núcleo es la

[6]La imagen fue tomada de la presentación del primer entregable del SYLLABUS-REFACTORY

parte en la que se enfoca la revisión de código, teniendo en cuenta los módulos liberados. Los módulos del núcleo son Componentes Educativos, Programa Académico, Estructura Curricular, Oferta Académica y Matriculación.

Los Componentes Educativos es una generalización de materias en diferentes niveles académicos y en sus diferentes formas de impartición. Por ejemplo un módulo es una materia que se imparte en postgrado, en pregrado en cambio existe actividad extracurricular y asignatura. Todos estos tipos actividades académicas las almacenará en un repositorio común. Un componente educativo es la unidad básica para impartir conocimiento hacia los profesionales en formación (Estudiante).

La configuración de Componentes Educativos permite a una secretaria de una escuela, Cittes o asociación configurar aspectos referentes a la forma de impartición de la asignatura dentro de una Unidad Operativa (Escuela) para un tipo de estudio (Pregrado, Postgrado, Formación Continua) y modalidad (Distancia, Presencial, Virtual). Una secretaria puede configurar: esquema de evaluación, duración, tipo de recursos que se utiliza, el mecanismo de calificación, si requiere material bibliográfico, componentes de la asignatura. Todos estos factores son distintos dependiendo del nivel académico y modalidad.

El Programa Académico (Carrera), es el programa de formación que se debe cursar para obtener un título académico. El proceso de creación de un programa académico parte desde la presentación del proyecto para la creación de dicha carrera hasta la aprobación por parte de un ente superior. Cada programa académico esta dado por una modalidad y un nivel de estudio, por lo tanto con cada Programa Académico se obtiene un título diferente, lo que implica también que cada Estructura Curricular sea diferente.

La Estructura Curricular es el conjunto de materias que debe cursar un estudiante para obtener un título profesional. Las asignaturas dependiendo del modo de aprobación del programa curricular tienen una diferente configuración. Por ejemplo si el modo de aprobación es por créditos, cada asignatura dependiendo del programa académico va a pertenecer a un grupo de créditos o a otro, en este modo de aprobación no hay una restricción para tomar las diversas asignaturas, lo que si se toma en cuenta es un número máximo de créditos que se puede tomar. Sí es modo de aprobación por asignaturas se va a tener un encadenamiento académico donde una asignatura es un requisito para tomar otra. El plan de estudios es creado por una escuela, pero debe ser aprobado previamente por un ente regulador de educación superior, en este caso el Conesup (Consejo Nacional de Educación Superior).

El proceso de Oferta Académica es previo al proceso de matriculación, el cual consiste en ofertar las asignaturas que pueden tomar los estudiantes.

La matrícula es el proceso de inscripción de un estudiante en las asignaturas que desee

tomar con el fin de aprobarlas y cumplir con su plan de estudio. La matrícula es el componente que tiene mayor interacción porque necesita toda la información brindada por los módulos que comprende esta primera liberación, por lo tanto para tener un proceso exitoso de matriculación todos los demás componentes deberán estar sincronizados y funcionando, porque la matriculación es un proceso con cierto grado de complejidad para el sistema.

3.2. Identificación de procesos.

En la tabla 2 se aprecia los módulos del SYLLABUS-REFACTORY, se puede observar los usuarios que se conectaran a este módulo, la dependencia de los módulos y la frecuencia de uso de cada modulo. En base a estos datos se determina el módulo o módulos críticos para el funcionamiento del sistema.

Tabla 2: *Módulos del sistema SYLLABUS-REFACTORY.*

	Matriculación	Oferta Académica	Componentes Educativos	Plan de Estudio	Estructura Curricular
Usuarios	Estudiante, Secretarias	Secretarias	Secretarias	Secretarias, Directores	Secretarias, Direcctores
Dependencia	Todos los módulos	Componentes Educativos, Plan de Estudio Estructura Curricular	Ninguno	Ninguno	Componentes educativos, Plan de estudio
Frecuencia de uso	Muy frecuente	Muy frecuente	Muy frecuente	Poco frecuente	Poco frecuente

El criterio para seleccionar los módulos del sistema SYLLABUS-REFACTORY para la revisión de código es por la fecha de la primera liberación del mismo, puesto que dichos módulos pertenecen al núcleo y son parte fundamental para la gestión académica de la UTPL.

Se estima que el módulo con mayor número de usuarios es el de matriculación, porque los estudiantes es el grupo más numeroso de usuarios del sistema, además este módulo depende de todos los módulos del sistema y el uso es muy frecuente. Por lo tanto la matriculación es un proceso crítico del sistema SYLLABUS-REFACTORY y es la razón de su existencia, por lo tanto al módulo de matriculación debe ser fundamental realizarle la comprobación de su seguridad.

3.2.1. Autenticación y Autorización.

Los procesos de autenticación como de autorización son parte fundamental de la seguridad de una aplicación, pero este tema no ha sido definido aun en el SYLLABUS-

REFACTORY[7], por lo tanto no se pudo hacer una revisión de este tema, se está trabajando con un usuario único que tiene acceso a todos los componentes. La autenticación actual es una simple comprobación donde el campo de usuario no debe ser nulo.

En la autenticación debe tomarse en cuenta las contraseñas de usuarios estén debidamente encriptadas y el acceso sea exclusivo para el administrador del sistema. La autenticación es la forma de saber que usuario es el que está ejecutando un proceso en el sistema.

La autenticación va de la mano de la autorización, en la lógica de negocio se está utilizando CSLA[8] , el cual tiene métodos definidos para realizar el proceso de autorización.

3.3. Comprobar qué tan seguros son los módulos seleccionados.

3.3.1. Revisiones de código de lógica de negocio (BL)[9]

La revisión de código es una buena técnica que ayuda a conseguir que se tenga un producto de calidad desde el punto de vista de la seguridad, por tal razón desde el inicio hay que hacer estas revisiones puesto que la seguridad no es casual, sino se va construyendo desde el inicio (Análisis) del proceso de desarrollo de software y se va fortaleciendo conforme avanza el proceso de desarrollo. La seguridad del software es parte de cada una de las etapas del ciclo de vida del mismo.

Es conocido que detectar errores en las primeras fases de desarrollo es más económico que detectar errores en una etapa avanzada del proceso, por esta razón se debe hacer una revisión exhaustiva, teniendo en cuenta buenas prácticas para escribir código. Por ejemplo:

- Sí existe código repetitivo es mejor refactorizar. Refactorice con frecuencia, esto ayuda a mantener el código organizado.
- Cuando se escriba un método este tiene que ser corto porque es fácil de entender, fácil de reutilizar, fácil de probar.
- Los nombres de las variables tienen que estar acorde a lo que se está haciendo, no nombrarlas de cualquier forma.
- Asignarle una sola responsabilidad a una clase.
- Crear lotes de pruebas (Plan de pruebas). El plan de pruebas para la revisión de código del SYLLABUS-REFACTORY se encuentra como anexo.
- Los comentarios son exclusivos para documentación.[10]

[7]Los módulos no han sido implementados hasta el momento en que se realizó la revisión.

[8]CSLA: http://www.lhotka.NET/cslanet/

[9]Las revisiones de código del presente trabajo están orientados al ambiente .NET, específicamente al lenguaje C#

[10]Obtenido en: http://msmvps.com/blogs/lopez/archive/2009/08/09/los-10-mandamientos-para-crear-buen-c-243-digo.aspxhacer

El núcleo se basa en poder ejecutar el proceso de matriculación básica de estudiantes. Para poder realizar la matriculación es necesario, poder crear asignaturas con su respectiva configuración, luego debe formarse una estructura curricular, de allí deben ser ofertadas las asignaturas por planes de estudio correspondientes, esto es por parte de una escuela, adicionalmente en la matrícula son los estudiantes los que van a inscribirse en las diversas asignaturas, en caso que no exista el estudiante registrado existe un subproceso de matrícula que permite registrar estudiantes, generalmente se da con estudiante nuevos.

La revisión de código para comprobar que estén validándose los datos de entrada se lo ha hecho a clases editables porque sus propiedades se persisten. Las propiedades de estas clases deben tener sus respectivas validaciones tanto longitud (tamaño del texto) como las de validaciones de caracteres permitidos con el fin de evitar inyecciones.

CSLA tiene dos grandes grupos de clases de negocio y son de solo lectura y editables cada una con sus diferentes estereotipos de clases de negocio. A los objetos de solo lectura solo se puede setear el valor de sus propiedades internamente, es decir dentro de la librería que contiene la clase, esto depende del nivel de acceso, pero externamente se puede únicamente obtener el valor seteado, además no se persisten las propiedades de clases de este tipo. Mientras que con los objetos editables se puede setear las propiedades públicas desde cualquier sitio y se persisten estos datos, es decir los objetos editables tienen un método "save()" a diferencia de los objetos de solo lectura, el cual permite el almacenamiento de datos en un repositorio previo cumplimiento de las reglas de negocio.[28]

Las clases criterio son un estereotipo de clases editables, pero sus propiedades no se persisten, lo que si ocurre con los otros estereotipos CSLA, las clases criterio son editables porque es necesario editar el valor de sus propiedades externamente y se necesita hacer la validación de propiedades. Los datos de clases criterio no se persisten pero van a la base de datos como parámetros de entrada para la obtención de registros.

3.3.2. Revisión al módulo de Componentes Educativos.

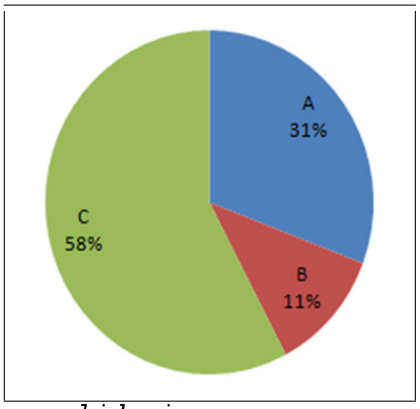
Este módulo permite la creación y configuración de asignaturas en los diferentes niveles de formación académica como es pregrado, postgrado y formación continua. Cada nivel tiene su respectiva configuración y sus respectivas asignaturas o módulos.

Este módulo permite la creación de asignaturas por parte de una secretaria de una escuela, en este módulo cada asignatura tiene las propiedades básicas, como es el código, nombre, descripción, fecha de creación, unidad propietaria (escuela), tipo de unidad (taller, asignatura), estado (activa o inactiva), área de formación (Técnica, Biológica, Administrativa, Socio-Humanística), cantidad de créditos.

El módulo Componentes Educativos permite hacer la configuración de asignaturas, por un tipo de aprobación (por asignaturas, por créditos), configuración de unidades admi-

nistradoras es decir ¿qué escuela va a dictar la asignatura? o ¿quién se encarga de toda la logística de esta asignatura? Este componente ya tiene un mayor grado de complejidad, está formado por varias clases, muchas de estas aún les falta validaciones a sus propiedades, otras clases en cambio están con validaciones incompletas.

En este módulo existe una gran cantidad de clases editables, para que sea posible la creación y configuración de componentes educativos que serán utilizados prácticamente en todos los otros módulos del sistema SYLLABUS-REFACTORY.



A: Clases que no contienen validaciones pero no poseen propiedades tipo string.
B: Clases con problemas de seguridad. No se valida los datos.
C: Clases sin problemas de seguridad.

Figura 2: *Revisión de código del módulo Componentes Educativos.*

Existe un total de 26 clases de negocio editables de las cuales un 11 % de clases poseen bugs de seguridad, debido a la falta de validaciones en sus datos, el 31 % son clases que no contienen validaciones pero estas clases no tienen propiedades tipo string, es decir, no se puede ingresar cadenas de texto, dando como resultado el 42 % de clases que no contienen validaciones. Esto se lo puede apreciar gráficamente en la Figura 2. Todas las clases que no contienen validaciones son un potencial riesgo para el sistema, porque a través de esta se puede ejecutar un ataque. Mientras que el 58 % de estas clases se encuentran validadas y sin errores.

3.3.3. Revisión al módulo Programa Académico y de Estructura Curricular.

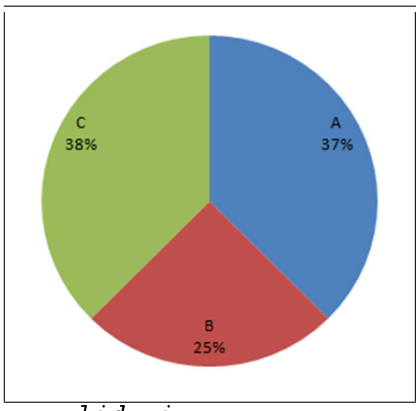
Debido a que los módulos Programa Académico y Estructura Curricular van de la mano y están en la misma librería se lo ha tomado como uno solo para la revisión de código de las clases de negocio, aunque a nivel de arquitectura sean diferentes paquetes.

Programa Académico: Es una carrera, la cual debe cursar un estudiante para obtener un título académico, este título puede ser de pregrado o postgrado en sus distintas modalidades. Un Programa Académico es diferente de otro aunque sea ofrecido en una misma

escuela. La modalidad es uno de los parámetros en que difiere cada Programa Académico, por lo tanto si una escuela ofrece una carrera en modalidad abierta y otra en modalidad clásica estas no son las mismas carreras. También el Programa Académico difiere en cada título profesional, en nivel de pregrado se ofrece un título de tercer nivel por ejemplo una Ingeniería, sí se trata de postgrado el título es de cuarto nivel.

Estructura curricular: Consiste en la configuración de un plan que debe cursar un alumno para obtener un título académico profesional, es decir agregar las asignaturas desde el repositorio de asignaturas hacia el plan correspondiente, dando lugar a la formación de la malla curricular que los estudiantes deberán aprobar para obtener su titulación.

En estos módulos existen clases que no tienen ninguna validación o están validadas parcialmente, otras clases serán eliminadas, se crearan otras debido a nuevas definiciones de negocio, también se modificarán las clases que ya están implementadas.



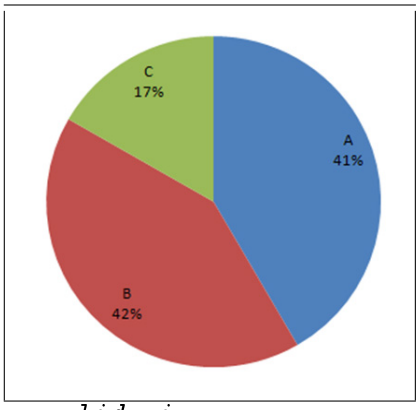
A: Clases que no contienen validaciones pero no poseen propiedades tipo string.
B: Clases con problemas de seguridad. No se valida los datos.
C: Clases sin problemas de seguridad.

Figura 3: *Revisión de código de los módulos Programa Académico y Estructura Curricular.*

Los módulos Programa Académico y Estructura Curricular tienen 16 clases editables de las cuales cuatro clases tienen bug de seguridad, es decir el 25 % de este módulo esta propenso a ataques debido a que las clases no poseen validaciones de los datos, mientras que el 37.5 % de las clases son potenciales riesgos porque estas clases no poseen validaciones de los datos de entada pero no poseen propiedades tipo string, El 37.5 % de las clases de este modulo son seguras debido a que se realiza la validación satisfactoria de todos los datos, por lo que no será posible ejecutar un ataque por estas clases de negocio. Esto se puede apreciarla en la Figura 3.

3.3.4. Revisión al módulo de Oferta académica.

El proceso de oferta académica se realiza previo al de matriculación, consiste en ofrecer las asignaturas a los estudiantes para que puedan completar su malla académica y obtener su título profesional. Quien ejecuta el proceso de oferta es la secretaria de cada escuela, previa aprobación de Dirección General Académica.



A: Clases que no contienen validaciones pero no poseen propiedades tipo string.
B: Clases con problemas de seguridad. No se valida los datos.
C: Clases sin problemas de seguridad.

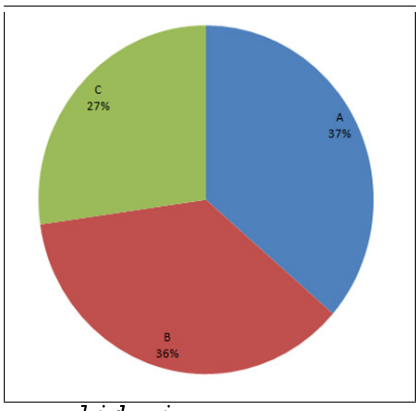
Figura 4: *Revisión del módulo Oferta Académica.*

El módulo Oferta Académica posee 12 clases editables de las cuales 5 clases tienen bugs de seguridad, que corresponde al 42 % de clases con errores, es un porcentaje alto, porque las pocas clases editables que tiene este módulo, los bug se dan por la falta de validaciones en las propiedades de estas clases. El 41 % de estas clases no poseen validaciones pero no contienen propiedades tipo string, por lo que en estas clases no se puede hacer un ataque, pero tampoco se realizan validaciones de los datos. El 17 % de estas clases son seguras, debido a que se realiza satisfactoriamente la validación de cada uno de los datos.

3.3.5. Revisión al módulo de Matrícula.

El proceso de matrícula consiste en la inscripción de estudiantes en las diferentes asignaturas que han ofertado las distintas escuelas, teniendo en cuenta que cada estudiante tiene un potencial de matrícula, este podrá matricularse en un cierto número de materia. El sistema mostrará las asignaturas que un estudiante puede tomar en base a ciertas reglas de negocio y de acuerdo al modo de aprobación que puede ser por asignaturas o por créditos, cada uno de estos modos de aprobación tiene sus respectivas reglas. Los estudiantes a matricularse que no se encuentren registrados, se registraran en el proceso de matriculación, esto es en el caso de estudiantes nuevos.

El módulo de Matrícula contiene 22 clases editables de las cuales 8 clases poseen bugs de seguridad que corresponde al 36 % de errores, todos los bugs son por falta de validaciones en las propiedades. El 37 % de las clases no poseen validaciones pero no poseen propiedades



A: Clases que no contienen validaciones pero no poseen propiedades tipo string.
B: Clases con problemas de seguridad. No se valida los datos.
C: Clases sin problemas de segur

Figura 5: *Revisión del módulo Matrícula*

string. El 27 % de estas clases son seguras debido a que se posee las validaciones de todas las propiedades. Este módulo será utilizado con mayor frecuencia porque existirá la matrícula en línea, por tal motivo estarán conectados a este módulo secretarias y estudiantes con el fin de realizar el proceso de matriculación, por lo que este módulo tiene una alta probabilidad de ataque.

3.3.6. Datos de la revisión.

Los datos obtenidos de la revisión de código se encuentran como anexos en la sección de revisión de lógica de negocio, donde está detallado cada una de las clases revisadas y los errores que se han encontrado.

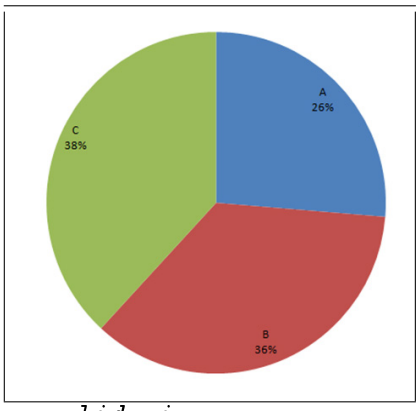
Se aprecia en la lógica de negocio que existe un 26.32 % de clases revisadas que contienen bugs de seguridad debido a que sus propiedades no tienen validaciones, este porcentaje corresponde a todas las clases que contienen propiedades del tipo string de las cuales algunas clases no contienen ninguna validación, otras están validadas parcialmente.

El módulo con mayor probabilidad de ataque es el módulo de Matrícula porque este módulo contiene un número elevado de bugs (36.36 % de sus clases), además es el módulo que va a tener mayor número de usuarios conectados porque va a existir la matrícula en línea, por lo que a este módulo hay que ponerle mayor énfasis en corregir sus errores.

Otro módulo que contiene con un número elevado de errores el de oferta académica, a pesar que contiene pocas clases editables su porcentaje de errores es alto con un 41.67 % de sus clases con errores. Este módulo será utilizado generalmente por secretarias pero eso no quiere decir que por este módulo no se ejecutarán ataques.

Existen clases que contienen propiedades del tipo numérico o fechas, estas clases no son

un peligro para la seguridad del sistema debido a que por estos campos no se pueden hacer inyecciones, pero estas clases igual deben contener validaciones. Las validaciones serán por cuestiones de negocio donde se valide rangos, valores permitidos, valores nulos, longitudes, etc.. Todas las propiedades son necesarias para el correcto funcionamiento del negocio por lo tanto todas las clases deben contener sus respectivas validaciones de negocio.



A: Clases que no contienen validaciones pero no poseen propiedades tipo string.
B: Clases con problemas de seguridad. No se valida los datos.
C: Clases sin problemas de seguridad.

Figura 6: *Revisión de código de la lógica de negocio.*

El total de clases de lógica de negocio que fueron revisadas son 76, de las cuales el 26 % de las clases no han sido validadas pero estas clases no contienen propiedades del tipo string. El 36 % de las clases tienen problemas de seguridad debido a la falta de validación de las propiedades tipo string, estas clases son propensas a ataques sobre todo de inyecciones debido a que sus propiedades pueden contener cualquier valor. El 38 % de las clases son totalmente seguras, son pocas clases que se encuentran con las debidas validaciones de datos. En la Figura 6 se puede apreciar estos datos.

En total existen un 62 % de clases que no poseen validaciones de datos, por algunas de estas se puede hacer ataque debido a la existencia de propiedades string en otras clases no porque solo poseen propiedades tipo numéricas o fechas, todas estas clases están incompletas y contienen bugs de seguridad por la falta de validaciones. Es un porcentaje realmente alto, pero este porcentaje tiende a la baja debido a que el sistema está en construcción y cada vez se van corrigiendo estos errores..

En las clases de lógica de negocio se hizo la revisión de clases editables porque a través de estas se pueden setear los valores a cada una de las propiedades para luego ser persistidas en la base de datos. Las clases criterio son clases editables que no persisten sus valores pero a través de estas clases se setea los criterios de búsqueda de listas o registros. En la

Tabla 3: *Revisión de código de la lógica de negocio.*

	Componentes Educativos	Estructura Curricular y Programa Académico	Oferta Académica	Matriculación
A	26	16	12	22
B	8	6	5	8
C	3	4	5	8
D	15	6	2	6
E	13	2	1	7

A: Número de clases editables.
B: Estas clases no contienen validaciones pero como no contienen propiedades por las que se pueda hacer una inyección. Pueden convertirse en potencial peligro.
C: Clases con problemas de seguridad.
D: Clases Clases seguras.
E: Clases Criterio (criterios de búsqueda).

Tabla 3 se observa un resumen de los datos de la revisión en cada uno de los módulos.

La revisión de código se hizo de forma manual debido a que las herramientas de revisión automática de código no realizan una revisión de propiedades que deben contener validaciones. Con el criterio de una persona experta se puede determinar que propiedades deben y cuáles no deben tener validaciones, ejemplo: las propiedades que no deben tener validaciones son las propiedades con el método set es internal, prívate o protect porque estos valores son seteados internamente por lo tanto ya son datos validos. Las propiedades que tienen el set público pueden ser accedidas y seteado su valor desde cualquier lugar, por lo tanto todas las propiedades de este tipo deben tener validaciones.

3.4. Revisión de código a la interfaz gráfica de usuario.

En la interfaz gráfica se debe realizar revisiones de código para verificar el manejo de excepciones, es decir validar el uso de los bloques try-catch. En el SYLLABUS-REFACTORY se aplica el manejo de excepciones a través del ELMAH[11], la cual consiste en capturar la excepción y con una llamada a un método, se guarda el error en una base de datos para luego ser visualizada y analizada por parte del administrador del sistema.

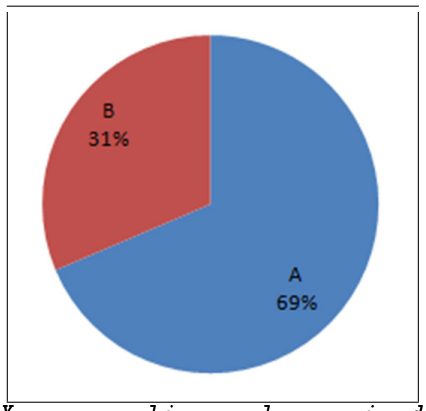
Para el manejo de errores es recomendable almacenar los errores generados por las excepciones con el fin de tener un historial, para ello es útil el ELMAH, este permite registrar los errores en una base de datos, los errores pueden ser visualizados posteriormente con el fin de tomar las decisiones pertinentes con respecto a cada excepción. El ELMAH tiene

[11]Mas información en: http://code.google.com/p/elmah/

soporte para base de datos SQLSERVER y ORACLE, además es una herramienta de código abierto, por lo tanto puede ser modificada y adaptarlo las necesidades de cualquier sistema.

El control de excepciones se lo debe hacer en los métodos lanzados por eventos (DirectEvents) o en métodos AJAX (DirectMethods) porque se encuentran en la capa superior de la aplicación, por lo tanto es allí donde se debe tratar las excepciones.

Durante la revisión de código a controles de usuario ascx y páginas Web aspx se verificó que no se realiza un correcto control y manejo de excepciones, en algunas páginas ningún método tiene un manejo de excepciones, otras en cambio todo está correctamente manejado, lo más común que se ha encontrado durante el proceso de revisión es que el método Page_Load no tiene un manejo de excepciones, mientras que en el resto de métodos se lo realiza correctamente.



A: Eventos y métodos AJAX que realizan el manejo de excepciones.
B: Eventos y métodos AJAX que no realizan el manejo de excepciones.

Figura 7: *Eventos y métodos AJAX de la interfaz gráfica de usuario.*

Existe un total de 121 eventos y métodos AJAX distribuidos en las diferentes páginas aspx, controles de usuario y controles personalizados de los cuales el 31 % no contiene el manejo de excepciones, mientras que en el 69 % restante se realiza correctamente la utilización de los bloques try-catch, muchos de estos errores es porque el método Page_Load no posee los respectivos bloques de manejo de excepciones. En la Figura 7 se observa gráficamente el porcentaje de métodos y eventos con manejo de excepciones.

En la Tabla 4 se observa los bugs encontrados por cada módulo con respecto al manejo de excepciones, es muy repetitivo el no realizar el manejo de excepciones o no se lo hace en el método Page_Load de una página aspx o un user control. En los anexos de la revisión de código de la interfaz gráfica de usuario se encuentran los datos de la revisión de las páginas y controles Web.

Tabla 4: *Revisión de código de la interfaz gráfica de usuario.*

	Componentes Educativos	Estructura Curricular y Programa Académico	Oferta Académica	Matriculación
A	2	4	3	3
B	5	9	8	10
C	0	4	0	0
D	29	35	32	25
E	15	10	8	5
F	4	5 (50 %) de errores	3	3
G	11	5	5	2

A: Número de páginas aspx.
B: Número de user controls.
C: Número de Web controls (controles personalizados).
D: Número de eventos y direct methods.
E: Bugs, No se hace manejo de excepciones.
F: Page_Load con manejo de excepciones.
G: Errores de procedimientos diferentes al Page_Load.
H: Páginas completaménte con manejo de excepciones.

Resharper y coderush: En esta revisión se hizo una validación de estándares y se verifica que no existan fallos en el producto. En el SYLLABUS-REFACTORY se realizó una revisión manual pero apoyándose en Resharper y en Coderush. Tanto en la lógica de negocio como en el code-behaind de la interfaz gráfica de usuario se utilizó estas herramientas para la agilizar la revisión de código.

Resharper se utilizó por la navegabilidad entre clase y la facilidad de buscar las mismas. Esta funcionalidad fue útil debido a que el sistema es demasiado grande, por lo tanto existen clases y archivos los cuales para acceder toman demasiado tiempo al menos buscándolos manualmente. El poder de Resharper está en el acceso al código para agilizar la revisión.

Coderush fue otra herramienta que se utilizó para la revisión de código debido a que este plugin tiene la funcionalidad de indentación de código con líneas de colores, lo cual es útil al momento de saber dónde empieza y dónde termina una método o un bloque de código. Sí se trata de pequeñas porciones de código cada bloque es fácil de entender, pero cuando los métodos son demasiado largos y tienen muchos ciclos, condiciones, etc. es útil la propiedad de indentación para poder entender de mejor manera el código fuente.

3.5. Revisión de código de base de datos (Inyecciones SQL).

Una mala práctica de desarrollo es formar una sentencia SQL en la capa de acceso a datos y enviarla a ejecutar en un repositorio de datos, esto ha llevado a que surjan las inyecciones SQL, donde, se forma una sentencia con parámetros que vienen desde una interfaz de usuario u otro sistema y si no tienen los datos validados se ejecutará una sentencia alterada en la base de datos. Una solución para evitar las inyecciones SQL es la utilización de procedimientos almacenados.

Actualmente la mayoría de bases de datos soportan los procedimientos almacenados, lo que es una ventaja si se quiere migrar de una a otra base de datos. Al utilizar procedimientos almacenados se evita en un buen porcentaje las inyecciones SQL. Hay que tener en cuenta que existen dos tipos de procedimientos almacenados, los dinámicos (no todas las bases de datos soportan) y estáticos.

Al utilizar procedimientos almacenados estáticos los parámetros que se envían hacia la base de datos se toman tal cual fueron enviados, por ejemplos si envía a guardar (insert o update) una cadena de texto similar a esto: “select * from tabla”, la cadena se guarda tal cual está y lo mismo ocurre al momento de recuperar la cadena.

Al utilizar procedimientos almacenados dinámicos se recomienda no enviar cadenas de texto, porque estas cadenas son propensas a cambios por consiguiente cambian el procedimiento. Al enviar una cadena de texto invalida a un procedimiento almacenado dinámico esta se concatena con otras cadenas, dando como resultado una inyección, lo cual es un grave problema de seguridad, no es recomendable enviar cadenas de texto, sino parámetros como números o fechas.

En el sistema SYLLABUS-REFACTORY se utilizan procedimientos almacenados estáticos, esto evita las inyecciones SQL, estos procedimientos son tanto insert, update, delete y select. Existen algunos procedimientos almacenados dinámicos, algunos de estos procedimientos reciben parámetros numéricos lo que evita el paso de caracteres extraños, otros reciben cadenas de texto.

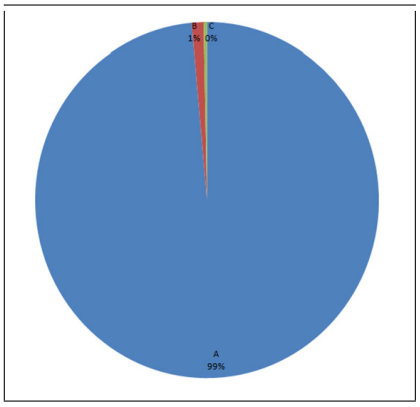
Se encontró un procedimiento en la base de datos que esta propenso a inyecciones SQL, inicialmente la clase de negocio que llamaba al procedimiento almacenado no tenía validaciones de sus propiedades lo que permitió ejecutar una inyección, pero posteriormente esta clase ya tenía su respectiva validación eliminando el bug de seguridad encontrado.

En el SYLLABUS-REFACTORY los procedimientos almacenados dinámicos son utilizados para hacer selecciones, debido a que hay sentencias complejas o repetitivas donde solo cambia una variable o una pequeña sección de código, por lo tanto al utilizar un procedimiento almacenado dinámico se elimina código repetitivo en la base de datos. En el SYLLABUS-REFACTORY no existen procedimientos almacenados dinámicos para hacer inserciones, actualizaciones o eliminaciones.

Los procedimientos almacenados dinámicos hay que utilizarlos cuando sea estrictamente necesario, así se evita problemas de seguridad. Cuando se utilice procedimientos almacenados dinámicos enviar datos numéricos o fechas, pero tratar de no enviar cadenas de texto, sí se envía una cadena de texto hay que cerciorarse que esta haya sido validada, para evitar inyecciones de cualquier tipo.

En los paquetes de administración del sistema el 4.41 % de los paquetes tienen problemas de inyección, pero todos estos paquetes son códigos de catálogos que utiliza el sistema en sus diferentes procesos, lo que implica que estos datos no son ingresados por los usuarios en la interfaz gráfica, sino estos códigos están en el sistema para cumplir con condiciones del negocio.

En la Figura 8 se observa que el 98.6 % de los paquetes son estáticos, mientras que el 0.003 % de los paquetes son dinámicos que no contienen parámetros de entrada tipo string y el 1.11 % de los paquetes son propensos a inyecciones porque son dinámicos y reciben como parámetros de entrada cadenas de texto.



A: Paquetes estáticos.
B: Paquetes dinámicos que reciben parámetros tipo cadenas de texto.
C: Paquetes dinámicos que no reciben parámetros tipo cadenas de texto.

Figura 8: *Revisión de Paquetes SGA(Sistema de Gestión Académica).*

En los paquetes del grupo SGA (Sistema de Gestión académica) el 1.11 % de los procedimientos almacenado son inseguros, pero solo se ha logrado hacer una inyección SQL exitosa en todo el sistema desde la interfaz gráfica de usuario, esta inyecciones se dio porque se conocía el código y las clases de negocio no tenían validaciones, por lo tanto validaciones tampoco estaban en la interfaz gráfica de usuario. En otras clases donde se utilizaba otros procedimientos almacenados dinámicos no se pudo hacer las inyecciones debido a que las clases de negocio que llamaban a estos procedimientos tenían validaciones, por lo tanto las clases de negocio y la interfaz gráfica no permitían el ingreso de caracteres extraños los cuales son necesarios para realizar una inyección.

En el único procedimiento almacenado por el que se logro hacer la inyección SQL fue PRC_OBT_VERIFICAR_CODIGO, este procedimiento almacenado permite verificar sí el valor de campo de una tabla específica existe o no existe, por lo general es utilizado para verificar los códigos de los registros, debido a una regla de negocio que dice "los códigos no pueden repetirse en una tabla". Este procedimiento almacenado es llamado desde una clase comando llamada VerificarRepetido. Una clase comando es un estereotipo CSLA que permite realizar una acción y devuelve un valor.

La sentencia está escrita en el procedimiento siguiente forma:

lc_query := 'SELECT COUNT('||ec_columna||') FROM '||ec_tabla|| ' WHERE LOWER('||ec_columna||') = LOWER(''||ec_valor||'');

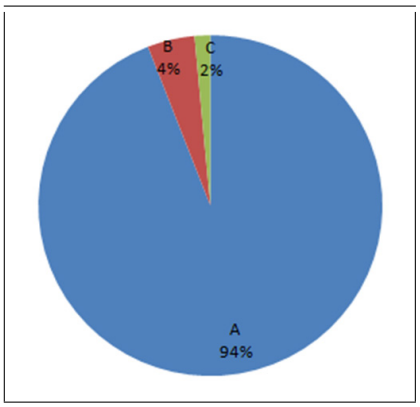
Se puede manipular al procedimiento almacenado debido a que no tiene ninguna validación en la clase de negocio, tanto de caracteres extraños como de longitud de la cadena, además el procedimiento es sencillo.

En el procedimiento PRC_OBT_AULAS se encontró una vulnerabilidad pero este procedimiento es tan complejo que cuando se intento hacer la inyección se generaban errores de ejecución por la sintaxis incorrecta del procedimiento, por tal motivo no se pudo inyectar a través de este procedimiento. Para ejecutar este procedimiento se ingresan datos por la interfaz gráfica de usuario, un parámetro es una cadena de texto que no tenía ninguna validación, por el que se podía ingresar cualquier valor.

Nota: El sistema está en construcción, posteriormente se quiso hacer inyecciones pero no fue posible porque en las clases de negocio se ha ido implementado las validaciones de datos.

En el grupo de paquetes administración (ADM) el 4 % de los paquetes son propensos a inyecciones debido a que estos son procedimientos almacenados dinámicos que contienen parámetros de entrada tipo texto. El 2 % de los paquetes son dinámicos que no reciben parámetros del tipo string, estos paquetes reciben números y fechas, por los cuales no es posible hacer ningún tipo de inyección. Como se lo aprecia en la Figura 9 el 94 % de los paquetes son estáticos, es decir por estos paquetes no será posible realizar inyecciones SQL.

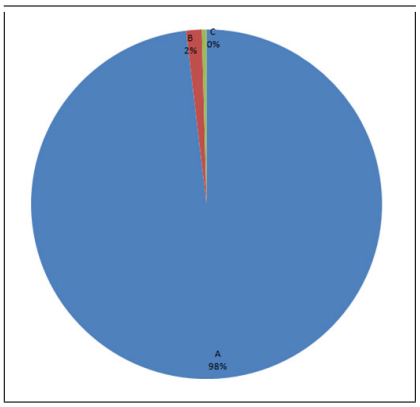
De todos los paquetes revisados el 1.44 % contienen bugs de seguridad para la base de datos, todos los bugs encontrados, representan un posible problema de seguridad en la base de datos, porque son vulnerabilidades donde se puede realizar inyecciones SQL. Todos los procedimientos almacenados dinámicos del SYLLABUS-REFACTORY son selecciones de datos, pero los propensos a inyecciones son los que reciben como parámetros datos varchar2, en algunos casos no se realizan validaciones en la aplicación. Las selecciones de datos que se realizan con procedimientos almacenados dinámicos son operaciones complejas que no se pueden realizar fácilmente, para evitar problemas de seguridad se deben



A: Paquetes estáticos.
B: Paquetes dinámicos que reciben parámetros tipo cadenas de texto.
C: Paquetes dinámicos que no reciben parámetros tipo cadenas de texto.

Figura 9: *Revisión de Paquetes ADM (Administración).*

validar los datos en la aplicación y enviar al procedimiento almacenado correspondiente. Los procedimientos almacenados del SYLLABUS-REFACTORY para realizar inserciones, actualizaciones y eliminaciones son procedimientos almacenados estáticos. Como se lo observa en la Figura 10 es mínima la cantidad de procedimientos almacenados dinámicos que se utilizan en el SYLLABUS-REFACTORY, la mayoría de los procedimientos almacenados son estáticos.



A: Paquetes estáticos.
B: Paquetes dinámicos que reciben parámetros tipo cadenas de texto.
C: Paquetes dinámicos que no reciben parámetros tipo cadenas de texto.

Figura 10: *Paquetes de base de datos.*

En la Tabla 5 se observa un resumen de los datos obtenidos de la revisión de procedimientos almacenados. Los procedimientos almacenados propensos a inyecciones son los dinámicos. En los anexos correspondientes a la revisión de paquetes de base de datos se encuentra detallada la información de los procedimientos revisados y en que paquetes se encuentran.

Tabla 5: *Revisión de paquetes de la base de datos.*

	ADM	SGA
Número de procedimientos	68	628
Procedimientos dinámicos	4	9
Procedimientos dinámicos sin peligro	1	2
Procedimientos dinámicos propensos a inyecciones	3	7
Procedimientos estáticos	64	619
Número de paquetes	10	111

TOAD: La revisión de código de paquetes en la base de datos se lo hizo con el TOAD, esta herramienta que sirve para la creación y ejecución de paquetes. También se realizaron inyecciones directamente en el navegador Web a través de la interfaz gráfica de usuario para observar el flujo de información.

La forma en la que se realizó las inyecciones fue primero desde el TOAD, ejecutado directamente en los procedimientos almacenados dinámicos, luego una vez hecha exitosa la inyección y obtenido una cadena para inyectar se lo hizo desde la aplicación, es decir desde un navegador Web. En la gran mayoría de los casos fue inútil la inyección debido a que las propiedades de las clases estaban validadas.

De los procedimientos almacenados dinámicos revisados solo en uno es en el que se pudo hacer una inyección SQL desde la interfaz gráfica de usuario debido a la falta de validación de datos. Con las inyecciones que se ejecutaron se pudo obtener y modificar los datos. Ventajosamente en la interfaz gráfica usuario donde se utiliza este procedimiento se está haciendo manejo de excepciones, en este caso sí la sentencia ingresada tiene errores de sintaxis se presenta un mensaje personalizado y no existirá fuga de información, por lo tanto no se puede verificar el motivo del error. En los procedimientos almacenados se observó lo siguiente:

- Por los paquetes del grupo SGA no se podrá hacer inyecciones SQL.
- Por los paquetes del grupo ADM se logro hacer una inyección SQL.
- Todos los procedimientos almacenados dinámicos del grupo ADM son selecciones de datos de las tablas de catálogos.
- Los datos que nos son códigos de catálogos están debidamente validados.
- Todas las selecciones de estos procedimientos almacenados son complejas.
- No se pueden hacer inyecciones desde la interfaz de usuario a estos paquetes.

- Debido a las validaciones que se tiene en las clases de negocio estos paquetes no representan problemas de seguridad al sistema.

3.6. Primeras Observaciones.

3.6.1. Validaciones.

En una aplicación Web para realizar revisiones de código enfocadas a la seguridad se debe tener en cuenta que las validaciones deben estar tanto en la lógica de negocio como en la interfaz gráfica de usuario. La validaciones en la interfaz gráfica de usuario son para mejorar el rendimiento de la aplicación y evitar viajes innecesarios al servidor consumiendo recursos tanto del servidor como de red, sí no se realiza la validación en el cliente por cualquier situación, por ejemplo el bloqueo de ejecución scripts, en este caso se hará la validación en la lógica de negocio.

En el SYLLABUS-REFACTORY se está utilizando X-VAL con Data Annotations para realizar las validaciones en la lógica de negocio, lo que permite verificar cuando un objeto es válido o invalido. Cuando un objeto es inválido no se permite realizar operaciones de persistencia de datos.

X-val[12] conjuntamente con Data Annotations[13] permiten realizar validaciones de datos de entrada y exportar las validaciones desde la lógica de negocio hacia la interfaz de usuario lo que conlleva a que el SYLLABUS-REFACTORY tenga las mismas validaciones tanto en el cliente como en el servidor. Todas las validaciones que están con Data Annotations se replican en la interfaz de usuario. También existen otro tipo de validaciones, que implican validaciones del negocio, estas son validaciones personalizadas que se las trata directamente en la lógica de negocio, en el caso que se las pueda controlar en el cliente se está generando un javascript para realizar esta validación.

Durante el proceso de revisión se encontró muchas clases de negocio que no tienen validaciones en sus propiedades, debido a que el sistema se encuentra en proceso de construcción, pero durante el proceso de revisión las clases que no estaban validadas se han ido corrigiendo.

En el caso de conectarse a una base de datos, antes de ejecutar un insert o un update o cualquier otra sentencia, lo primero que debe hacerse es cerciorarse que el objeto a ser persistido sea válido, es decir debe cumplir con todas las validaciones tanto de negocio como las de seguridad. El SYLLABUS-REFACTORY está cumpliendo con este requeri-

[12]Ma información sobre X-Val en: http://xval.codeplex.com/
[13]En: http://www.codeproject.com/KB/validation/Data_Annotations_with_ASP.aspx

miento, pero no en su totalidad. Todo objeto es validado pero como hay propiedades que no tienen su validación, estas propiedades son propensas a inyecciones.

Las validaciones en el cliente se lo hace a través de scripts (javascript, vscript), pero en los navegadores existe la opción para desactivar su ejecución, lo que implica que no se ejecuten las validaciones en el cliente, por lo tanto se envían datos inválidos al servidor, es por esta razón que la validación de datos debe hacerse también en la lógica de negocio, no siempre llegarán datos correctos y validados al servidor. Se debe tener en cuenta que en el SYLLABUS-REFACTORY las validaciones están en la lógica de negocio y de allí se las exporta a la interfaz de usuario, lo que conlleva a que la validación de los objetos siempre se la realice en la lógica de negocio y no importa que el navegador tenga desactivado la ejecución de script, porque los datos siempre van a ser validados, la persistencia es segura y se evita todo tipo de inyecciones.

Para poder ejecutar una página Web creada en EXTNET es necesario tener habilitada la opción de ejecución de scripts, de no ser así las páginas Web no se cargarán, porque EXTNET genera scripts para la creación de la página Web en el cliente. Las validaciones siempre se van a ejecutar en el cliente por el hecho que se necesita tener habilitada la opción de ejecución de script.

3.6.2. Paso de parámetros.

El envío de parámetros por las urls debe ser evitado, porque sí se conoce una dirección simplemente se la escribe en el navegador y se accede fácilmente a una página que ha sido abierta previamente. Se puede acceder a una página del sistema sin necesidad de logeo o acceder a una página de acceso restringido para el rol con el que se conecta el usuario, esto se da porque no se tiene un correcto manejo tanto de autenticación como de autorización.

En el SYLLABUS-REFACTORY existe una única dirección para acceder al portal, lo que implica tener una url, esto es gracias a las propiedades que tiene EXTNET el cual es la plataforma que se está utilizando para las interfaces gráficas de usuario. Todas las páginas se abren dentro de un contenedor en la página principal, las url específicas no son visibles para los usuarios. La páginas aspx se despliegan dentro del contenedor principal, por lo tanto no se guardan las urls de cada página en el navegador.

Por las urls es posible hacer inyecciones, en caso que se envíe parámetros por este medio, el parámetro puede ser un criterio de búsqueda que va a una base de datos para obtener cierta información. Se debe evitar el envió de parámetros por urls, es preferible utilizar un método post o la variable de sesión, pero si se la utiliza no se debe abusar de esta porque consume memoria del servidor, por lo que se recomienda utilizarla solo para almacenar pequeños datos y no almacenar grandes objetos, al almacenar grandes objetos en sesión el servidor no avanzaría a responder todas las peticiones hechas por los clientes, dando

como consecuencia la negación de servicios y (o) una lenta respuesta a cada una de las peticiones.

Las inyecciones por intermedio de urls no van a ser posible en el SYLLABUS-REFACTORY, debido a que existe una única url, el paso de parámetros entre páginas se lo realiza por el código que está detrás de la página (code-behaind) o por intermedio del método post, también se utiliza la variable sesión pero se está guardando objetos grandes en algunos casos lo que no es recomendable.

3.6.3. Validaciones de valores nulos.

Se debe hacer comprobaciones con valores nulos en la entrada de datos, es decir los valores que reciben los métodos, sí no se puede recibir un valor nulo en un método se debe lanzar una excepción, caso contrario, el método podrá recibir estos valores.

Los valores nulos están siendo controlados con el fin de evitar problemas posteriores, son pocos los casos donde se permite valores nulos en el SYLLABUS-REFACTORY, pero estos son tratados de forma correcta, no se ha encontrado ningún inconveniente porque existe una correcta validación. Los datos nulos son utilizados para búsquedas, cuando el objeto es nulo se busca todos los registros o ninguno, dependiendo del caso.

3.6.4. Propiedades y métodos.

Se debe chequear todos los métodos y propiedades públicas que tenga una clase, porque lo público es lo que se expone hacia fuera del componente y por donde se puede realizar un ataque. Lo privado, internal o protegido no necesitan una comprobación porque el propio objeto es el encargado de llamar a los métodos o propiedades con estos niéveles de acceso. Los parámetros que son enviados a métodos privados, protegidos o internal ya han sido validados previamente o son obtenidos de un repositorio de datos, por lo tanto son datos seguros, porque antes de guardar un objeto este debe ser válido.

Cuando se realice un proceso interno en la lógica de negocio los datos ya deben estar validados, sí los datos son erróneos los objetos deben ser inválidos. Las clases CSLA permiten crear objetos de negocio que realizan todo este proceso, se invalidan los objetos de negocio. CSLA no permiten que se realicen operaciones mientras los objetos no cumplan con las validaciones. CSLA permite también realizar validaciones con mayor grado de complejidad, validaciones como verificar si un objeto ya existe, lo que implica obtener datos del repositorio y realizar la operación respectiva. En el caso que existan validaciones complejas primero deben pasar las validaciones estándar para luego ser ejecutadas las validaciones complejas.

Las validaciones de seguridad son repetitivas en la mayoría de las clases de negocio. Las validaciones más comunes son: el control de caracteres extraños en cadenas de texto, campo es obligatorio, longitud de una cadena.

Se debe verificar que los métodos no reciban parámetros inválidos, siempre se debe hacer esta comprobación por cada parámetro o al menos cuando se reciba un dato del tipo string. Sí se recibe varios parámetros en un método es mejor refactorizar y recibir un objeto que contenga todas las propiedades que anteriormente se tenía como parámetros y comprobar que todo el objeto sea válido. Todos las clases de negocio CSLA heredan la clase BuisnessBase, por lo tanto tienen la propiedad IsValid y gracias a Data Annotations y X-VAL, las propiedades son validadas, sí el objeto es válido se realizan las operaciones respectivas.

En CSLA se maneja los objetos criterio, estos se los utiliza para el envío de parámetros a métodos de una clase, sí el objeto criterio es válido se realizarán las operaciones respectivas. Generalmente los objetos criterio son utilizados para realizar búsquedas, este objeto contendrá los parámetros de búsqueda para obtener los objetos requeridos. Los objetos criterio no tienen validaciones complejas, tienen validaciones estándar como comprobación de caracteres extraños, rangos de valores etc.. Los objetos criterio pueden tener complejas en caso que se lo requiera.

La revisión de validaciones debe hacerse a propiedades públicas de las clases, cada propiedad debe tener su propia validación, en el caso de cadenas de texto se debe validar que no contengan carácter que permitan hacer una inyección, caracteres como: ‘, “, <, >, estos caracteres permiten concatenar o modificar cadenas de código.

En el SYLLABUS-REFACTORY realizan validaciones estándar para: el ingreso de valores como caracteres extraños, valores requeridos, rangos de números, etc., pero no todas las propiedades tienen validaciones. En una segunda revisión realizada en algunas clases que no tenían validaciones estas han sido validadas, lo que quiere decir que el sistema cada vez tiene mayor grado de seguridad.

Se debe hacer revisiones de pruebas unitarias hechas por desarrolladores, porque ello da la pauta para saber qué ha hecho el desarrollador, dando la primera visión general de qué puede estar mal. En las pruebas unitarias se debe comprobar las validaciones, si estas están correctas se puede dar por comprobada una clase, caso contrario se debe tomar las medidas correspondientes con el fin de evitar cualquier problema futuro.

3.6.5. Comentarios.

Se recomienda una correcta aplicación de comentarios a clases, métodos y propiedades. Los comentarios deben ser explicativos y cortos, para no dar falsas interpretaciones al código, en caso que se necesite hacer cambios, sí este está mal comentado se pueden realizar cambios erróneos. Los comentarios tienen que ser oportunos y tampoco se debe comentar de más. Los comentarios son para documentación e interpretación de código.

En el SYLLABUS-REFACTORY existen algunos comentarios que están erróneamente aplicados y como se trata de una aplicación que está en proceso de construcción pueden darse cambios incorrectos en el código por parte de un desarrollador diferente al que creó el código original. Una técnica que se está aplicando en el SYLLABUS-REFACTORY es poner los nombres de los desarrolladores y fecha que han creado y modificado las clases, lo que da lugar a tener una asesoría del uso de las clases por parte de los desarrolladores en caso que se lo requiera. En el SYLLABUS-REFACTORY existen equipos de desarrollo auto organizados, los desarrolladores conocen el código que otros han desarrollado, esto es gracias a la técnica de desarrollo en parejas, parte de esta técnica es que un desarrollador revisa el código de otro, de esta forma el conocimiento no queda en una sola persona, sino que se comparte a todo el equipo dando lugar a independencia entre sus miembros.

3.6.6. Manejo de excepciones.

En cuanto manejo de excepciones, esto se lo debe hacer en la capa superior del sistema, si ocurre una excepción en una capas inferiores por ejemplo en la capa de acceso a datos, la excepción deberá ser lanzada a una capa superior, en este caso a la lógica de negocio y de la lógica de negocio se lanzará a la interfaz gráfica de usuario. El error debe ser tratado en la capa de mayor nivel por ejemplo en el code-behaind de una página aspx o de un servicio Web.

La utilización del ELMAH es una buena alternativa para tener historial de excepciones ocurridas. Este framework permite guardar los errores por lo tanto se pueden verificar cuáles han ocurrido y darles la respectiva solución. Esto es útil en caso que no pueda hacerse un debug.

3.6.7. Cookies.

En el sistema SYLLABUS-REFACTORY no se utiliza cookies. Es una gran ventaja para la seguridad porque por este medio se puede ingresar datos inyectados a la aplicación, por tal motivo existe un elemento menos por el que se debe preocupar.

En muchas aplicaciones se da mal uso a las cookies, estas deben ser únicamente para la personalización de páginas por parte de los usuarios, no se deben almacenar datos confidenciales como contraseñas. Los datos almacenados en cookies no deben comprometer a la seguridad del sistema. Se recomienda almacenar datos como productos de una canasta de compras, colores para la visualización personalizada de páginas Web, etc.

En el SYLLABUS-REFACTORY la forma de almacenar y acceder a los datos que están en el cliente se lo hace por intermedio de stores. Un store es un pequeño almacén de datos de EXTNET, el cual permite mantener los datos en el cliente y accederlos cada vez que sea necesario.

3.6.8. Archivo de configuración.

En cuanto al archivo de configuración es allí donde se almacenan las claves de acceso a datos, para ello se debe encriptar esta información. Otro error que se comete cuando se pone a producción un sistema es el no cambiar a false la propiedad debug de la etiqueta compilation

<compilation debug="true"></compilation>

porque al estar en true esta propiedad se puede lanzar un error y presentará información del archivo de configuración, lo que es fuga de información. En caso de que no se tenga encriptada la clave de acceso al repositorio de datos, se podrá visualizar fácilmente, que es muy sensible.

3.7. ¿Qué tan seguro es el SYLLABUS-REFACTORY?

El tiempo que tomó la revisión de todos los módulos fue de dos meses y medio. El trabajo fue realizado por una persona. Se hizo la revisión de código en cada una de las capas de la aplicación, donde en cada capa se realizo las respectivas pruebas, con el fin de conocer que problemas de seguridad tiene el sistema. A nivel de base de datos se realizo la revisión de cada uno de los procedimientos almacenados, con el fin de ejecutar inyecciones SQL. En la lógica de negocio se verificó que todos los datos que se ingresen tengan su respectiva validación, con el fin de evitar todo tipo de inyecciones. En la capa de presentación ser verifico el correcto manejo de excepciones, con el fin de evitar fugas de información, además se hizo pruebas para verificar que no existan inyecciones XSS.

Ningún sistema es 100 % seguro, cada vez hay nuevas formas de atacarlo. El objetivo de realizar una revisión de código enfocado a la seguridad es disminuir la posibilidad de penetraciones. Durante la revisión de código al SYLLABUS-REFACTORY se han tomado en cuenta los siguientes puntos:

- El framework de interfaz gráfica de usuario no es propenso a inyecciones XSS, además ASP.NET tiene soporte para detectar este tipo inyecciones, por lo tanto es invulnerable ante ataques XSS.
- En la interfaz gráfica de usuario no se hace el manejo de excepciones en todos los bloques de código correspondiente.
- En la lógica de negocio existe la validación de datos, pero no se encuentran todos los datos de entrada validados.
- Se está utilizando procedimientos almacenados en la base de datos, por lo que disminuye la posibilidad de inyecciones SQL, pero existen los procedimientos almacenados dinámicos, los cuales pueden ser vulnerables a este tipo de ataques.
- Solo una inyección SQL se ejecutó exitosamente a través de la interfaz gráfica de usuario hacia un procedimiento almacenado dinámico.

- Se hicieron otras inyecciones pero debido a validaciones de datos no se ejecutaron exitosamente.

Gracias a la revisión de código realizada a los diferentes módulos se determina que el SYLLABUS-REFACTORY es INSEGURO porque:

- La lógica de negocio es insegura en 26.32 % debido a la falta de validaciones en las propiedades públicas.
- La base de datos es insegura en un 1.44 % debido a los paquetes dinámicos propensos a inyecciones.
- La interfaz gráfica de usuario es insegura en 31.4 % debido a la falta de manejo de excepciones.

El porcentaje de clases de negocio que falta validaciones es alto. No se está cumpliendo a cabalidad con la regla que dice: "todo dato de entrada debe ser validado".

La falta de validaciones de datos en la capa de negocio hace que la aplicación sea propensa a inyecciones. En el caso de los procedimientos almacenados dinámicos que reciben cadenas de texto que no están validadas hace que la aplicación sea propensa a inyecciones SQL, esto sucedió con el procedimiento PRC_OBT_VERIFICAR_CODIGO que es llamado desde la clase de negocio VerificarRepetido la cual no tiene validaciones de los datos de entrada.

No se podrá hacer inyecciones XSS a la aplicación porque se está utilizando EXTNET y ASP.NET, tampoco se podrá deshabilitar la opción de ejecución de script en estas páginas, porque EXTNET genera scripts que son necesarios para la construcción de la página Web, sí tiene deshabilitada esta opción no se cargará la página Web, por lo tanto no se podrá hacer uso de la aplicación. El tener habilitada la opción de ejecución de script permite ejecutar siempre las validaciones en el cliente, por tal motivo se va a optimizar recursos, dando a la aplicación mayor velocidad y seguridad.

La mayoría de procedimientos almacenados del SYLLABUS-REFACTORY son estáticos, son pocos los procedimientos almacenados dinámicos. Los bug encontrados en la revisión de base de datos son por procedimientos almacenados dinámicos que reciben cadenas de texto, esto se corrige con la validación de datos de entrada en la lógica de negocio.

La falta de manejo de excepciones en la interfaz gráfica de usuario es el error que más se comete, por lo tanto el sistema es propenso a fugas de información. A pesar que el SYLLABUS-REFACTORY tiene el ELMAH como framework para el manejo de excepciones, no se lo utiliza a cabalidad. En muchos DirectMethods y DirectEvents hacen falta los bloques try-catch, por lo tanto tampoco se guardan las excepciones.

Se debe corregir cada uno de los errores que se encuentren en cada capa del sistema. Cada error es un potencial peligro para la seguridad del sistema.

3.8. Observaciones.

Un error cometido inicialmente durante la revisión de código fue revisar que en la interfaz gráfica de usuario se estén validando los datos cuando lo que se debía verificar era la lógica de negocio, las validaciones realizadas en el interfaz de usuario es por cuestiones de optimización de recursos y no hacer viajes innecesarios al servidor, el cual por lo general tienen muchas peticiones. El validar a un objeto que ya ha pasado por una validación previa hace que la validación en el servidor sea para comprobar que el objeto este correcto y no devolverlo para modificar datos que están erróneos, haciendo en vano todo el procesamiento realizado.

Otro error cometido durante el proceso fue el no guardar la fecha conjuntamente con el bug encontrado, esto es útil con el fin de tener un historial de bugs encontrados.

Inicialmente no se conocía que se iba a revisar exactamente, conforme se avanza el proceso se va adquiriendo la capacidad de discernir donde están los errores. La corrección de cada error deber ser propia del desarrollador, con el fin que no vuelva a comerte los mismos errores.

4. Capítulo IV. Contrarrestar los problemas de seguridad. ¿Cómo Evitar los problemas de seguridad?

En el presente capítulo se propone una solución a los problemas de seguridad del software con el fin de evitar agujeros de seguridad, por los cuales se pueden ejecutar penetraciones indebidas. Además se realiza un estándar para determinar el nivel de seguridad en aplicaciones de negocio similares.

Todo bug encontrado es un potencial riesgo para la seguridad del sistema, por esa razón una vez encontrado un bug debe solucionase antes que se convierta en un error grave. Las penetraciones a los sistemas de información se dan debido a que no se corrigen errores, en ocasiones los errores son por negligencia o porque simplemente no se ha realizado una revisión de código o sí se realiza una revisión de código es para verificar estándares y requerimientos funcionales. Pocas veces se realiza una revisión de código con el fin de encontrar vulnerabilidades de seguridad durante el proceso de desarrollo de software.

Es imposible tener un sistema impenetrable al cien por ciento, debido a que cada vez surgen nuevos problemas de seguridad, para ello se debe estar preparado y evitar cualquier ataque. A los sistemas se les deben realizar varias pruebas, no solo de funcionalidad sino también de seguridad con el fin de no tener margen de error.

4.1. Revisiones de código.

Las revisiones de código es la forma con mayor grado de eficiencia para encontrar bugs en los sistemas, a través de esta se verifica sí está haciéndo una correcta validación de datos de entrada, se realiza un correcto manejo de excepciones, se encripta ciertos datos, etc.. Una revisión de código debe realizarse de forma exhaustiva, línea por línea, porque no se sabe donde pueda haber un problema de seguridad.

Las herramientas automáticas ayudan a agilizar la revisión de código, pero estas herramientas no tienen la heurística necesaria para definir donde debe ir un bloque try-catch-finalize, donde se debe hacer un control de valores nulos o donde hay que hacer validaciones de datos.

4.1.1. Excepciones no controladas.

Las excepciones son acciones imprevistas que pueden ocurrir durante la ejecución de un proceso del sistema. Un correcto manejo de excepciones evita fugas de información del sistema. Cuando ocurre un error y no se realiza un correcto manejo a la excepción se deja abierto para un ataque, porque se muestra información delicada del sistema. La única forma comprobar que se está realizando el manejo de excepciones de forma correcta es por medio de una revisión de código manual.

4.1.2. Evitar inyecciones.

El ingreso de datos desde la interfaz gráfica de usuario es la fuente principal para las inyecciones, al estar validados los datos de entrada se puede evitar las inyecciones de todo tipo. Por lo tanto todo dato de entrada debe ser validado, para ello existen frameworks para cumplir este requerimiento.

Para tener un mayor grado de seguridad se debe evitar formar sentencias SQL en la aplicación, porque estas son susceptibles de cambio y fáciles de hacer una inyección. Esta práctica es común en algunos desarrolladores, porque muchas veces no se tiene conocimiento de los problemas de seguridad que se pueden generar.

Se deben crear políticas para la creación de procedimientos almacenados y ejecución de sentencias SQL en una base de datos. Existen dos tipos de procedimientos almacenados, dinámicos y estáticos. Con los procedimientos estáticos no se puede hacer inyecciones SQL, porque ejecuta las sentencias tal cual fueron creadas, sí los parámetros que se reciben son cadenas de texto con código inyectado estas no modificarán la sentencia.

Los procedimientos almacenados dinámicos son propensos a modificación de la sentencia, lo que da lugar a inyecciones, por tal motivo debe evitarse en lo posible este tipo de procedimientos. Sí se utiliza procedimientos almacenados dinámicos, no se debe recibir parámetros con cadenas de texto abierto, sino recibir cadenas de texto validadas.

Una forma de evitar cualquier tipo de inyección es la validación de todos los datos de entrada en la aplicación, porque es por medio de la interfaz gráfica de usuario donde se dan los mayores problemas de seguridad. En campos abiertos de textos es por donde se hace el ingreso de inyecciones ya sea SQL, XSS u otros.

Las validaciones no solo se las debe hacer en el cliente o capa de presentación sino también en el servidor o capa de negocio. En el cliente se realiza validaciones por medio de javascript, estas validaciones deben ser las mismas que están el servidor. Muchos navegadores para optimizar su rendimiento tienen la opción de deshabilitar la ejecución de scripts lo que implica que no se ejecuten las validaciones en el cliente, por tal motivo al servidor pueden ir datos inválidos lo que conlleva a ejecutar las validaciones en la lógica de negocio para asegurar que los datos son correctos.

El motivo de tener validaciones en la capa de presentación es por optimización de recursos y evitar viajes innecesarios al servidor, evitando procesamiento que muchas veces es solo para mostrar un mensaje de error que puede ser mostrado directamente en el cliente.

Se debe tener claro que la interfaz gráfica de usuario no es el único medio para el ingreso de datos. Se debe hacer un correcto uso de cookies, estas no deben contener información para los procesos del sistema, sino únicamente información básica que no involucren datos confidenciales.

Se pueden hacer inyecciones por urls, las inyecciones por este medio se dan cuando se pasan parámetros los cuales son visibles en la url. El paso de parámetros se lo debe hacer por el code-behaind, por el método post o la variable sesión.

4.1.3. Contraseñas

Una contraseña es muy importante para la identificación de usuarios en el sistema, por ello el manejo de contraseñas debe ser de forma apropiada. Es recomendable seguir siguientes las normas:

- El almacén de contraseñas debe estar encriptado. Así no podrá ser vista por cualquier persona, estas no deben ser vistas ni por el administrador del sistema.
- Una contraseña no debe ser un nombre o una frase porque es fácil de descifrar.
- Las contraseñas deben contener combinaciones alfanuméricas y algunos caracteres especiales permitidos como: @, $, %, etc. de esta forma será difícil de descifrarla.
- Se deben cambiar constantemente las claves, al menos las de administración.
- Es recomendable verificar que tan segura es una combinación de caracteres, para ello existe programas o algoritmos que validan la longitud y las combinaciones alfanumericas.

4.1.4. Autorización y autenticación.

Cuando se habla de seguridad estos métodos son los más comunes para comprobar el correcto ingreso de cada uno de los usuarios al sistema. Existen varios formas de realizar la autenticación y autorización, por ejemplo con Active Directory, la autenticación por formularios, con servicio LDap, otros.

Los procesos de autenticación son parte de la lógica de negocio. CSLA tiene soporte para el manejo de autenticación a través de formularios, active directory. CSLA además de tener soporte para los frameworks mencionados anteriormente también tiene sus propias clases e interfaces para la implementación de CSLAAutentication la cual se puede adicionada en el respectivo archivo de configuración con la siguiente línea:

<add key="CslaAuthentication" value="Csla" />

Es una ventaja que CSLA tenga soporte para varios frameworks de autenticación, debido a que da flexibilidad al sistema, porque se puede cambiar la forma de autenticación sin tener grandes problemas de impacto, los cambios serán tranparentes en otras capas de la aplicación. CSLA en cada clase de negocio tiene los métodos para verificar la autorización a cada proceso que se ejecute.

Autenticación: Es el proceso para identificar a los usuarios que se conectan al sistema, por lo general este proceso se lo hace a través de un nombre de usuario y una contraseña. El nombre de usuario debe ser único.

Autorización: Es el proceso que permite verificar qué funcionalidades de negocio puede ejecutar cada uno de los usuarios, debido a que en el negocio todos los usuarios no pueden ejecutar todas las funcionalidades, cada usuario está restringido a ejecutar las funcionalidades correspondientes a su rol.

A continuación se nombran algunas herramientas que ayudan a realizar la autenticación y autenticación:

- CSLA[14] es un framework para el manejo se lógica de negocio, pero tiene incorporado métodos que sirven para la autenticación y autorización. CSLA también tiene soporte para Active Directory.
- Active Directory[15] es una plataforma de manejo de directorios en una red distribuida Microsoft, el cual provee un servicio para el manejo de identificación que sirve tanto para la autorización y autenticación.
- Enterprise Library[16] es un framework que facilita el desarrollo de aplicación empresariales, el cual cuenta con un bloque llamado Security Application Block el cual cuenta con los métodos necesario tanto para autenticación con para la autorización.
- Servicio LDAP[17], el cual se basa en el acceso por roles de usuario, es decir con cada rol tiene acceso a ciertas funcionalidades que no se tiene con otro rol.

4.2. Informe de los bugs encontrados.

Una vez encontrado un bug de seguridad lo primero que debe hacer es informar a quien corresponda. El hacer conocer dónde están los errores es importante, porque se tomarán los correctivos necesarios para eliminar los problemas correspondientes. Si se habla de seguridad todo bug tiene prioridad alta, porque son puntos de acceso no autorizados a la aplicación.

Los bugs por lo general se tratan de errores en el código, por lo que se debe informar en que clases o archivos se ha encontrado los errores. La precisión al informar donde se encuentran los errores es muy importante para su corrección, porque con ello se logra optimizar el tiempo de corrección, el desarrollador ira exactamente al lugar del error y lo corregirá.

4.3. ¿Qué se debe revisar?

[14]CSLA: http://www.lhotka.NET/cslanet/
[15]www.microsoft.com/windowsserver2003/technologies/directory/activedirectory/default.mspx
[16]Entreprice Library: http://msdn.microsoft.com/en-us/library/ff648951.aspx
[17]LDAP: http://www.openldap.org/

Un sistema grande por lo general está dividido en capas, lo más común es tener tres capas: lógica de acceso a datos, lógica de negocio e interfaz gráfica de usuario. La revisión de código no se debe enfocar en una sola capa, cada capa tiene una responsabilidad específica.

Capa de acceso a datos: Esta capa debe encargarse de ejecutar sentencias SQL y procedimientos almacenados. En esta capa se realizan operaciones con los datos, todos los datos ya deben estar validados, porque ya han pasado por la lógica de negocio. Una de las cosas que se debe validar es que únicamente se haga referencia a la capa de acceso a datos desde la lógica de negocio con el fin de asegurar que todos los datos sean validos y cumplan con las reglas de negocio.

Base de Datos: Una forma de evitar problemas de seguridad como inyecciones SQL es utilizar procedimientos almacenados estáticos. No olvidar que existen dos tipos de procedimientos almacenados, los dinámicos y los estáticos. Los procedimientos dinámicos tienen su grado de inseguridad cuándo se envían parámetros de tipo texto. Para poder ejecutar una inyección se debe formar sentencias correctas y debidamente estructuras, de no ser así simplemente se lanzará una excepción de base de datos.

Los procedimientos almacenados dinámicos deben ser utilizados para hacer selecciones, debido a que hay sentencias complejas o repetitivas donde solo cambia una variable o una pequeña sección de código, por lo tanto al utilizar un procedimiento almacenado dinámico se elimina código repetitivo de la base de datos. Pueden existen procedimientos almacenados dinámicos para hacer inserciones, actualizaciones o eliminaciones, pero no es muy aconsejable tenerlos.

En las validaciones de datos se debe controlar el tamaño de cadenas texto, sí el tamaño es corto será difícil hacer una inyección por ejemplo con diez caracteres una sentencia podría quedar incompleta. También, no se debe permite caracteres extraños, así simplemente no se podrá hacer inyecciones.

Lógica de negocio: En esta capa existen validaciones propias del negocio, las cuales se verifica que cumplan con las reglas que dicta el mismo. Las reglas del negocio están enfocadas a la responsabilidad que cumple cada objeto de negocio con el fin de satisfacer los requerimientos funcionales.

Existen validaciones técnicas de datos, donde se verifica que las propiedades de clases cumplan con ciertas reglas de carácter técnico como: permitir o no el ingreso de ciertos caracteres, longitud de cadenas de texto, las cadenas de texto cumplan con reglas dadas por expresiones regulares, etc.. Todos estos son requerimientos no funcionales que no son dados por los usuarios, pero se los debe cumplir con el fin de dar al producto un valor agregado.

Las propiedades de clases editables de negocio tienen el método get y set público, por lo tanto se puede setear los valores desde cualquier lugar de donde se haga referencia, se debe tener validaciones técnicas por cada una de las propiedades con la finalidad de asegurar que los datos sean validados y evitar problemas de seguridad.

Las clases criterio son editables, a pesar que sus propiedades no se persisten, pero estas deben tener la respectiva validación, porque estos datos irán a un repositorio como criterios de búsqueda para hacer las selecciones y obtener una lista de registros.

Capa de presentación: Es la capa superior de una aplicación, esta es visualizada por usuarios finales. Con esta capa interactúan directamente los usuarios u otros sistemas (en el caso de un servicio Web) La capa de presentación hace referencia a la interfaz gráfica de usuario o servicios que ofrece a otros sistemas, para el intercambio de información y uso de la aplicación.

Es recomendable utilizar un framework para la interfaz gráfica de usuario, no solo para realizar interfaces gráficas elegantes, sino también para poder optimizar los recursos que tiene el cliente gracias al soporte de javascript que existe en los diferentes frameworks.

En la capa de presentación es donde debe realizarse el manejo de excepciones con el fin de evitar las fugas de información. El manejo de excepciones no se lo debe hacer en todos los métodos, sino solo en métodos que son llamados por eventos de la interfaz gráfica de usuario o por los métodos AJAX (DirectMethod).

4.4. ¿Qué se debe hacer?

4.4.1. Plan de pruebas

Para ejecutar la revisión de la seguridad un sistema lo primero que se debe hacer es un plan de pruebas, donde se detalla cada uno de los procesos que se ejecutarán para verificar que tan seguro es un sistema. Este plan contendrá detalladamente:

- Resumen.
- El o los objetivos de las pruebas.
- El enfoque del plan (En este caso está enfocado a la seguridad).
- Los roles de las personas que intervienen en la revisión.
- Las expectativas que se tiene de la revisión.
- Documentación que se requiera.
- Los requerimientos de los recursos.
- Reportes de bugs.
- Cronograma.
- Riesgos y dependencias
- Terminología usada.

4.4.2. Revisión de código.

Las herramientas automáticas para la revisión de código están enfocadas a las revisiones de estándares de programación como nombramiento de variables, comentarios (en clases, métodos y propiedades), optimización de código, etc., pero al momento de hacer revisiones de código enfocadas a la seguridad estas herramientas no son de mucha utilidad, motivo por el cual debe realizarse una revisión manual, pero apoyándose en herramientas automáticas para optimizar el tiempo de revisión.

En la lógica de negocio debe revisarse validaciones en propiedades públicas. Toda propiedad debe contener su respectiva validación. Es importante que las propiedades contengan validaciones, a través de propiedades tipo string se puede ingresar texto abierto, lo que permite hacer inyecciones de distinta índole.

En la capa de interfaz gráfica de usuario en el code-behaind se debe revisar la correcta aplicación de manejo de excepciones, este manejo se lo debe hacer en los métodos llamados por eventos o en métodos AJAX, estos son llamados desde javascripts. La razón por la que se debe hacer el manejo de excepciones en este tipo de métodos es para evitar el anidando a los bloques try-catch, es suficiente con un solo bloque try-catch en la parte superior de la aplicación, es decir en un evento o un método AJAX.

A la capa de acceso a datos se debe llegar únicamente a través de la lógica de negocio, por lo que implica tener datos validos al momento de ejecutar una sentencia en la base de datos. Pero la base es propensa a inyecciones SQL.

La mejor forma de evitar la inyección de SQL es teniendo procedimientos almacenados estáticos, sí existen procedimientos almacenados dinámicos se debe evitar parámetros tipo cadenas de texto, por lo tanto se debe verificar que los procedimientos almacenados dinámicos sean llamados desde clases donde sus propiedades hayan sido validadas para evitar inyecciones SQL.

4.4.3. Corrección de bugs. Ciclo de vida de un bug de seguridad.

En seguridad todo bug tiene alta prioridad, siempre deben ser corregidos todos los bugs que se descubran durante las revisiones de código. Por cada bug encontrado se debe seguir un proceso hasta corregir todos los bugs. A continuación se detalla el proceso para la corrección de un bug en un sistema informático:

1. El proceso se inicia con la revisión de código por parte del tester (revisor de código), este proceso consiste en ir detectando los bugs que pueden afectar a la seguridad del sistema.
2. Cada bug encontrado debe ser reportado individualmente al equipo de desarrollo con el fin de corregirlo. Cada bug es un problema diferente.

3. El equipo de desarrollo se encargará de corregir los bugs de seguridad una vez reportado el mismo.
4. Una vez corregido el bug el equipo de desarrollo de encarga de informar al equipo de testadores que la corrección ha sido ejecutada.
5. Luego el testeador se encarga de verificar que la revisión haya sido exitosa.
   a) Sí la revisión ha sido correcta el bug queda cerrado.
   b) De ser la revisión incorrecta la se vuelve a reportar el bug para su respectiva corrección. Se vuelve al paso 2.
6. Una vez corregido el error se da por finalizado el issue.

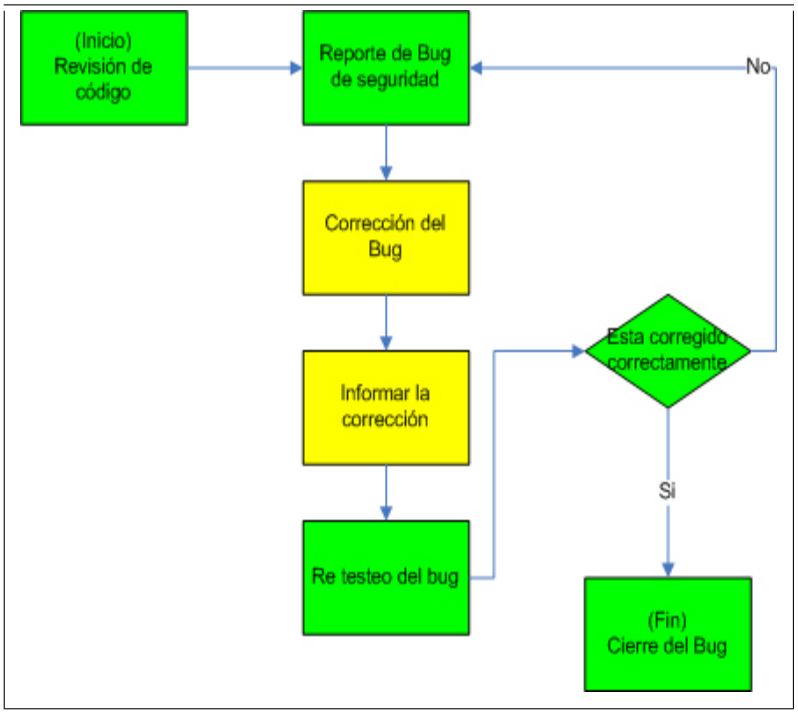


Figura 11: *Ciclo de vida de un bug de seguridad.*

En la Figura 11 se puede observar gráficamente el ciclo de vida de un bug, donde lo que está de color verde es lo que realiza el tester o revisor de código, mientras lo que esta de color amarillo es lo que realiza el equipo de desarrollo. El equipo de desarrollo debe ser independiente del equipo de tester.

4.5. ¿Qué validaciones hacer?

Parte del éxito en tener un código seguro es encontrar los bug del sistema y reportarlos a quien corresponda, esto es con el fin de que sean corregidos. Todo bug deberá ser corregido, porque se puede convertir en un riesgo potencial para el sistema.

4.5.1. En la interfaz gráfica de usuario.

La interfaz gráfica de usuario es lo que visualizan los usuarios finales, esta debe ser fácil de utilizar, tener mensajes claros, los nombres de las etiquetas entendibles. En la interfaz gráfica de usuario se debe verificar lo siguiente:

- El manejo de excepciones se lo debe hacer métodos llamados por eventos o en métodos AJAX.
- Que se estén guardando las excepciones en un repositorio de datos.
- Al usuario se muestren mensajes personalizados cuando se lancen excepciones, nunca se deben mostrar los mensajes de error lanzados por el sistema, esto es fuga de información.
- Correcta utilización de mensajes personalizados. No se deberá mostrar al usuario mensajes que lo confundan.
- Verificar que las validaciones sean exportadas a la interfaz gráfica de usuario y funcionen de igual forma como lo hacen en la lógica de negocio.
- Debe existir una única página de autenticación al sistema.
- Utilizar el método post, la variable sesión o el code-behaind para el paso de parámetros entre páginas Web. No utilizar la variable sesión para este fin.

4.5.2. En la lógica de negocio.

En la lógica de negocio es donde se encuentran las reglas que rigen el negocio, esta parte del sistema tiene que acoplarse a cambios, porque es lo más cambiante. En la lógica de negocio se debe validar lo siguiente:

- La autorización a los diversos módulos del sistema debe ser para los usuarios que tengan el privilegio de ingresar a los mismos.
- El proceso de autenticación es propio de la lógica de negocio.
- Cada clase de negocio debe tener un proceso de autorización.
- Todos los datos de entrada deben ser validados. La validación debe estar en cada una de las propiedades de las clases de lógica de negocio.
- Verificar que los métodos públicos reciban datos que hayan sido validados.
- El manejo de valores nulos. Existen casos donde se permiten valores nulos, en otros no. En los casos donde no se permitan valores nulos se debe lanzar una excepción, esta deberá ser capturada en la capa superior.
- No es necesario validar propiedades de solo lectura, porque estas son seteadas internamente, por lo general son datos recuperado de un repositorio, estos datos ya han sido validos previamente.

4.5.3. En la capa de acceso a datos y base de datos.

La capa de acceso a datos se encarga de la manipulación de los datos que se encuentran en un repositorio. Lo que se debe validar en esta capa es lo siguiente:

- Las contraseñas deben estar debidamente encriptadas.
- Los procedimientos almacenados dinámicos en lo posible no deben contener parámetros del tipo cadenas de texto. En el caso que tengan cadenas de texto como parámetros de entrada verificar que estén validadas.
- No se debe formar sentencias SQL en la aplicación, se debe hacer uso de procedimientos almacenados. Actualmente la gran mayoría de proveedores de bases de datos soporta procedimientos almacenados, incluso las herramientas exprés y las gratuitas.
- Tener una base de datos de excepciones, con el fin de tener un historial y poder hacer revisiones posteriores. También deberá existir la lógica para administrar el manejo de excepciones.

4.5.4. Pruebas unitarias para la validación de cadenas de texto.

Es recomendable realizar pruebas unitarias por parte de los desarrolladores, para verificar la validación de datos. La prueba deben ser para: validación de caracteres extraños, longitud de cadenas de texto, permitir valores nulos. Las pruebas unitarias es lo primero que se debe validar al momento de hacer una revisión de código, porque sí se encuentra una prueba fallida se lo debe notificar.

Como se observa en el Código 1, en la prueba unitaria primero se crea una clase abstracta con un objeto valido, luego se crea las diversas pruebas para cada uno de los parámetros, cada prueba es una clase que hereda de la clase abstracta, donde se le da valores validos e inválidos a cada una de las propiedades para observar qué pruebas son exitosas y cuáles no, con esto se comprueba que el objeto de negocio está cumpliendo con todas las validaciones de seguridad.

Las pruebas unitarias son para verificar que se cumplan las reglas de negocio plasmado en el código de la lógica de negocio, con estas pruebas se puede contrastar que no se permitan inyecciones de ningún tipo. La prueba unitaria que se encuentra en el Algoritmo 1 es una especificación. Esta especificación es una es un ejemplo de las validaciones que se deben hacer por cada propiedad tipo cadena de texto que se tenga en una clase de negocio.

Las pruebas unitarias pueden ser ejecutadas con Resharper, CodeRush, MSpec, entre otros, estas herramientas tienen un framework para poder ejecutar este tipo de pruebas.

Código 1 *Pruebas unitarias que siempre se deben realizar.*

```
//Creación de un objeto valido
public abstract class contexto_valido {
    protected static MiClase objMiClase;
    Establish context = () => {
        objMiClase = new MiClase();
        objMiClase.Nombre="Nombre Valido";
        objMiClase.Descripcion="Descripción valida del objeto";
    };
}
//Validación del nombre, validación correcta.
[Subject("MiClase")]
public class cuando_se_ingrese_el_Nombre_Valido : contexto_valido{
    Because of = () => objMiClase.Nombre = " Nombre Valido";
    ThenIt deberia_ser_valida = () => objMiClase.IsValid.ShouldBeTrue();
}
//Validación del nombre, no se debe permitir ingresar caracteres para una inyección.
[Subject("MiClase")]
public class cuando_se_ingrese_caracteres_extraños : contexto_valido{
     Because of = () => objMiClase.Nombre = " '); Select * from tadm_item;";
     ThenIt deberia_ser_invalida = () => objMiClase.IsValid.ShouldBeFalse();
}
//Se debe tener un limite de longitud de cadenas de texto donde X es el limite
[Subject("MiClase")]
public class cuando_se_ingrese_cadenas_de_longitud_mayor_a_X : contexto_valido{
    Because of = () => objMiClase.Nombre = UtilSpec.GenerarCadenaAlfaNumerica(X);
    ThenIt debe_ser_invalida = () => objMiClase.IsValid.ShouldBeFalse();
}
//Se debe tener un cotrol de valores nulos
[Subject("MiClase")]
public class Cuando_sea_nulo : contexto_valido{
    Because of = () => objMiClase.Nombre = null;
    ThenIt debe_ser_invalida = () => objMiClase.IsValid.ShouldBeFalse();
}
```

Observación.

El estándar propuesto puede ser aplicado a muchos sistemas empresariales, porque todos los sistemas afrontan los mismos problemas de seguridad y las soluciones son las mismas. Existen herramientas similares que se puede utilizar en otros lenguajes de programación. Las herramientas que utilice cada tester para revisión de código queda a su criterio, lo importante es detectar y corregir las vulnerabilidades para tener un producto de alta calidad y SEGURO.

5. Conclusiones y recomendaciones.

Conclusiones:

- De acuerdo con la revisión de código que se ha realizado al sistema de Gestión Académica de la Universidad Técnica Particular de Loja SYLLABUS-REFACTORY se concluye que el sistema presenta problemas de seguridad en los módulos revisados, debido a la falta de validaciones en los datos de entrada, utilización de procedimientos almacenados con sentencias SQL dinámicas y falta de manejo de excepciones, lo cual produce fuga de información a través de la interfaz gráfica de usuario e inyecciones SQL.

- Los componentes del sistema donde se encontraron errores de seguridad en sus respectivas clases son Oferta Académica en un 42 %, Matriculación en un 36 %, Programa Académico y Estructura Curricular en un 25 %, Componentes Educativos en un 11 %.

- Las causas por las que existen vulnerabilidades en el sistema en gran parte se dan debido al desconocimiento de la existencia de dichos problemas, es decir, de cómo se generan y como se solucionan, además, aun no se tiene definidos los respectivos requerimientos de seguridad del sistema.

- Las herramientas de revisión de código ayudaron a agilizar el proceso de revisión manual de código del sistema SYLLABUS-REFACTORY en un 30 %, es decir, el tiempo se redujo en una tercera parte en relación de ejecutar todo el proceso manualmente.

- Un procedimiento almacenado fue propenso a inyecciones SQL en toda la base de datos, debido a que contenía sentencias SQL dinámicas y no existía validaciones de los parámetros de entrada, el resto de procedimientos almacenados fue inmune a inyecciones SQL debido a que sus sentencias eran estáticas.

Recomendaciones:

- Para tener un sistema de información seguro se propone seguir el estándar planteado en el capítulo IV, para ejecutar revisiones de código en cada una de las capas de una aplicación.
- Se recomienda que un programador experto realice la revisión de código, porque fácilmente puede encontrar errores en el código y dar una recomendación adecuada para tener una fácil solución a los bugs encontrados.
- En proyectos de similares características se sugiere utilizar las herramientas planteadas en el presente proyecto, porque se adaptan al proceso de revisión manual de código.
- Una práctica recomendable que se realiza en el proyecto SYLLABUS-REFACTORY, consiste en poner el nombre de la persona que modifico una clase o archivo y además es un breve comentario con el motivo de la modificación.
- Cuando se integren nuevos desarrolladores, los problemas de seguridad deben ser socializados con la mayor prontitud para evitar que se reincida en errores ya corregidos por el equipo de desarrollo.
- En proyectos de características similares al SYLLABUS-REFACTORY se sugiere el uso de herramientas como Team Fundation la cual permite hacer el seguimiento para la corrección de los bugs encontrados y guardar el historial de los mismos. Además ayuda a mantener el código actualizado, con el fin de no duplicar trabajo en código que ha sido modificado.

Referencias

[1] F. Nick, "La seguridad en la ingeniería de software," Enero 2010.

[2] M. MINGUET, "IntroducciÓn a la seguridad informÁtica," Enero 2010.

[3] B. D. y. D. J. Lodderstedt T, "Secureuml: A uml-based modeling language for model-driven security?,"

[4] B. J. Posadas Juan, Simó José, "Un modelo para el desarrollo de aplicaciones distribuidas. servidor de comunicaciones," enero 2010.

[5] A. A. Areitio Javier, "Test de penetración y gestión de vulnerabilidades, estrategia clave para evaluar la seguridad de red," enero 2010.

[6] W. Mark, "Host discovery with nmap," November 2002.

[7] B. Fernando, "Una propuesta de seguridad en la información," Marzo 2010.

[8] T. V. Julio, *Derecho informático*, vol. Vol. 1. McGraw-Hill, tercera ed., 2004.

[9] L. M. S. Ruwase Olatunji, "A practical dynamic buffer overflow detector," 2007.

[10] R. Alexander, "Exception handling in object oriented systems: Towards emerging application areas and new programming paradigms," Julio 2003.

[11] P. Juan, "Manejo de errores usando excepciones java," Abril 2007.

[12] M. D. Palacios Daniel, "C sharp: Clases, polimorfismo y excepciones," Mayo 2002.

[13] S. S. y Jean Harrold Mary, "Analysis and testing of programs with exception-handling constructs," septiembre 2008.

[14] C. Roman, "Inyecciones sql," octubre 2008. [En línea] Disponible en: [http://www.romancortes.com/blog/inyecciones-sql/].

[15] F. Pete, "Sql injection and oracle, part one," Noviembre 2010.

[16] K. K. y. H. M. Kosuga Yuji, "Sania: Syntactic and semantic analysis for automated testing against sql injection," Mayo 2010.

[17] G. Jeremiah, "Cross-site scripting worms & viruses," Junio 2007.

[18] S. Kevin, "Are your web applications vulnerable?," 2005.

[19] C. Cesar, "Manipulating microsoft sql server using sql injection," Abril 2010.

[20] M. D. A. PÚBLICAS, "Aplicaciones utilizadas para el ejercicio de potestades," Junio 2004. [En línea] Disponible en: [http://publicaciones.administracion.es/].

[21] G. A. D. A. y García Navarro Juan F, "Mecanismos de seguridad de la informaciÓn en aplicaciones web," mayo 2006.

[22] ReSharper, "Developer productivity tool for microsoft visual studio," enero 2010. [En línea] Disponible en: [http://www.jetbrains.com/resharper/].

[23] DevExpress, "Coderush and refactor! pro downloads," enero 2010. [En línea] Disponible en: [http://www.devexpress.com/Downloads/Visual_S tudio_Add-in/].

[24] Microsoft, "Fxcop," 2010. [En línea] Disponible en: [http://msdn.microsoft.com/en-us/library/bb429476

[25] Microsoft, "Stylecop. [en línea] disponible en: [http://code.msdn.microsoft.com/sourceanalysis lease/projectreleases.aspx?release id=1425]," 2008.

[26] Q. Software, "Toad world," mayo 2010.

[27] C. J. A., "Source code review of the diebold voting system," julio 2007.

[28] R. Lhotka, *Expert C SHARP 2008 Business Objects*. Springer-Verlag New York, 2009.

[29] Anónimo, "Manejo de excepciones y errores en una arquitectura java," Abril 2009.

[30] B. Carmen, "Recursos para la educaciÓn estadÍstica en internet," Diciembre 2008.

[31] A. Chris, "Advanced sql injection in sql server applications," Marzo 2002.

[32] F. S. Consulting, "Fg-injector framework documentation," 2006.

[33] B. Cristian, "Seguridad fisica," Diciembre 2009.

[34] ELMAH, "Elmah. error logging modules and handlers for asp.net," 2010. [En línea] Disponible en: [http://code.google.com/p/elmah/].

[35] B. Fabrice, "Cross-site scripting (xss)," 2007.

[36] F. Gil, "Building asp.net validator using data annotations," Julio 2010. [En línea] Disponible en: [http://www.codeproject.com/KB/validation/Data_ Annotations_with_ASP.aspx].

[37] H4CK1T, "Desbordamiento de enteros en java," 2007.

[38] V. J. y. A. O. Halfond William, "A classification of sql injection attacks and countermeasures," Abril 2010.

[39] S. P. Injector, "Sql power injector. product information," 2007.

[40] S. Jarrod, "How to write insecure code," Diciembre 2009.

[41] C. Jerónimo, "Tratamiento de excepciones," Enero 2010.

[42] M. M. y Lam Monica S., "Automatic generation of xss and sql injection attacks with goal-directed model checking," Mayo 2010.

[43] Microsoft, "Windows server 2003 active directory," 2010. [En línea] Disponible en: [http://www.microsoft.com/windowsserver2003/tech nologies/directory/activedirectory/default.mspx].

[44] Microsoft, "System.componentmodel.dataannotations namespace," 2010. [En línea] Disponible en: [http://msdn.microsoft.com/en-us/library/system.componentmodel.dataannotations .aspx].

[45] Microsoft, "Microsoft enterprise library," 2010. [En línea] Disponible en: [http://msdn.microsoft.com/en-us/library/ff648951.aspx].

[46] S. P. y. S. D. Nadji Yacin, "Document structure integrity: A robust basis for cross-site scripting defense," Mayo 2010.

[47] Nmap, "Nmap," Diciembre 2009.

[48] S. Otto, "Cross site scripting," 2004.

[49] J. B. J. S. y. J. D. P. Barlet, H. Pujol, "Sistema de detección de anomalías de red ponencias basado en monitorización y predicción de tráfico," Enero 2010.

[50] B. Pablo, "Revisión de resharper 3.0 de jetbrains," Febrero 2008.

[51] G. Pablo, "Programación segura: Desbordamientos del búfer," Diciembre 2009.

[52] D. C. y. L. J. Romanovsky Alexander, "Handling in object oriented systems: Towards emerging application areas and new programming paradigms," Julio 2003.

[53] P. R. S, *Ingenieria del Software, Un enfoque practico.* McGraw-Hill, 2006.

[54] Sencha, "Sencha," 2010. [En línea] Disponible en: [http://www.sencha.com/].

[55] T. Sergio, "Seguridad física," Enero 2010.

[56] M. O. y Shulman Amichai, "Sql injection signatures evasion," Abril 2004.

[57] S. Steve, "xval," Febrero 2009. [En línea] Disponible en: [http://xval.codeplex.com/].

[58] C. Steven, "A web developers guide to cross-site scripting," enero 2003.

[59] J. N. K. E. K. C. V. G. Vogt Philipp, Nentwich Florian, "Cross-site scripting prevention with dynamic data tainting and static analysis," Marzo 2010.

6. Anexos

Plan de Pruebas
Syllabus Refactory

Versión [1.0.0]

Revisión y Tabla Final

Revisiones

Archivo	Método o Propiedad	Error	Observación

Características del Documento

Ítem	Detalles
Titulo del Documento	Plan de Pruebas
SUBTÍTULO:	Syllabus Refactory
Archivo:	Plan de Pruebas
Autor	Junior Sinche
Fecha de creación	2010-05-31
Ultima actualización	2010-06-04
Versión:	[1.0.0]
Estado:	Borrador

Tabla de Contenidos

Resumen

El Plan de Pruebas ha sido diseñado para verificar la seguridad del sistema de gestión académica Syllabus, partiendo del hecho que todo sistema es inseguro con el fin de encontrar posibles Bugs de seguridad en el código fuente y evitar cualquier infiltración al sistema.

Este plan describe cómo implementar o llevar a la práctica las estrategias que se usará para las pruebas, la organización y el manejo del proyecto. Este documento identifica: objetivos de prueba, metodologías y herramientas, resultados esperados, responsabilidades y recursos requeridos.

El equipo de Pruebas es responsable de la creación del plan de pruebas. Este documento ofrece una guía de las estrategias que el equipo usará para la solución de las pruebas. Es responsable de poner las expectativas de calidad e incorporarlas en el plan de pruebas. El responsable directo de la creación es del Tesista.

Objetivos

- Revisión del código de la lógica de negocio para encontrar propiedades no validadas, con el fin de evitar distintas inyecciones(SQL, XSS, etc..).
- Revisiones de código verificar el correcto Manejo de excepciones.
- Revisiones de Procedimientos almacenados para evitar inyecciones SQL.
- Ejecución de casos de pruebas.
- Identificar y reportar bug de seguridad.

Enfoque de pruebas

- **Pruebas de Seguridad (Alcance).**
  El objetivo de las pruebas es verificar que tan seguro es el sistema de gestión Académica Syllabus partiendo del hecho que todo sistema es inseguro.

- **Revisiones de código.**
  El objetivo de la revisión de código es validar el código fuente verificado que esté cumpliendo con estándares de seguridad como la validación de datos de entrada, correcto manejo de excepciones, y los datos que lleguen a los procedimientos almacenados sean validos con el fin de evitar inyecciones.

- **Ejecución de distintos tipos de inyecciones.**
  Realizar distintos tipo de inyecciones con el fin de encontrar las vulnerabilidades del sistema. Existen campos abiertos los cuales permiten el ingreso de cualquier dato por lo general las áreas de texto donde se escribe una descripción del contexto en el que estamos, estos campos son los propensos a inyecciones. Las inyecciones a realizar serán SQL, scripting, XML, HTML, etc.

Roles y responsabilidades de las pruebas

Recursos Humanos		
Rol	**Recursos Mínimos Recomendados.**	**Responsabilidades Específicas**
Director de tesis		Brindar supervisión en la dirección del proyecto, las responsabilidades incluyen: ▪ Planeación logística. ▪ Identifica motivaciones para realizar el proyecto. ▪ Apoya los intereses del test. ▪ Evalúa la efectividad del test.
Codirector de Tesis		Apoyar al Director de tesis. ▪ Coordinación con el director de tesis para que se cumplan los objetivos.

Recursos Humanos		
Rol	**Recursos Mínimos Recomendados.**	**Responsabilidades Específicas**
Tester (Tesista)		Implementa y ejecuta las pruebas. Las responsabilidades incluyen: ▪ Implementa una prueba como un conjunto de pruebas ▪ Ejecuta las colecciones de tests. ▪ Presenta y analiza los resultados del log. ▪ Analiza y recupera los fracasos del test. ▪ Documenta. ▪ Define la arquitectura de automatización de test. ▪ Verifica las técnicas de la prueba. ▪ Analiza y Evaluar los resultados

Expectativas de los resultados de las pruebas

- Que la aplicación reporte la menor cantidad de bugs posibles
- Que se documente el proceso de pruebas del sistema
- Que se generen indicadores del proceso de estabilización
- Que las pruebas ayuden a determinar el nivel seguridad en tiempo de desarrollo.

Documentación Requerida

Documentación de pruebas

- Reportes periódicos del estado de las pruebas
- Resumen de los resultados de las pruebas
- Documentación final de la liberación
- Plan de Pruebas
- Casos de Prueba (Inyecciones)
- Reporte de Issue

Datos de pruebas

Se requiere contar con el código fuente y tener acceso a los distintos servidores (TFS, Base de datos) con el fin de poder ejecutar la aplicación de una forma correcta, para encontrar las posibles vulnerabilidades.

Interacción con otras áreas

Se interactúa principalmente con el área de:

- **Desarrollo**
  Esta área es de vital importancia ya que es con quien se coordina la corrección de bugs reportados.

Procedimientos de pruebas

- **Revisión de código**.
  Consiste verificar que el código cumpla con estándares de seguridad, que va desde el correcto manejo de excepciones, la validación de todos los datos de entrada, y encontrar posible puertas traseras. La revisión de código de acuerdo a la capa a la que está verificando la seguridad, es diferente la revisión.

- **Inyecciones**
  Consiste en ejecutar inyecciones de diferentes tipos, y que la aplicación no se vea afectada por los posibles problemas que pueda causar esto.

- **Documentar e informar los bugs encontrados**
  Consiste en documentar e informar los bugs encontrados al equipo de desarrollo con el fin de que las vulnerabilidades encentradas sean tratadas de la mejor manera.

Revisión de pruebas

La coordinación de la revisión de las pruebas las realizará el director de tesis, codirector de tesis y con cada uno de los involucrados en el proceso.

Estado de la Estabilización y reportes

- En primer lugar definido el alcance y de acuerdo a las fechas de liberación se emitirá un reporte con los casos de pruebas diseñados.
- Luego durante la fase de estabilización, se emitirá un reporte semanal sobre el avance del proceso incluyendo issues reportados con sus respectivos estados.
- Se incluye un reporte con los bugs encontrados durante la revisión ejecutada.
- Documento de liberación de versión.

Requerimientos de los recursos de pruebas

Recursos

Hardware	Computadora Servidor para Base de Datos Infraestructura de Red
Software	Visual Studio Team Fundation Microsoft Excel Microsoft Word Utilitarios Oracle 10g Resharper CodeRush Express Style Cop TOAD
Datos	Código fuente.

Dotación de personal & Necesidades de entrenamiento.

Roles	Mínimo de recursos recomendados	Mínimo de recursos recomendados
Director y codirector de Tesis	Responsable de supervisar validar el proceso.	1
Tesista	Responsable de evaluar las condiciones para el proceso de pruebas. Responsable de diseñar los escenarios de pruebas y casos de pruebas. Responsable de ejecutar las pruebas. Responsable de documentar los resultados.	1

Reporte de Issues Herramientas y Métodos

Durante la ejecución de los casos de pruebas se irán reportando los issues respectivos del resultado de las pruebas los mismos que se clasifica como:

- Vulnerabilidad: Es un error que indica que la aplicación no cumple con alguna especificación de seguridad la cual es un posible punto de acceso no autorizado al sistema.

Decisión de Estrategia

Todas las vulnerabilidades son posibles puntos de acceso al sistema por lo que todos los bugs deben ser tratados con igual prioridad, ya que los módulos que se están analizando en los procesos críticos del sistema.

Cronograma

Riesgos y Dependencias

Riesgo	Prioridad	Responsable
Las reglas de negocio no estén bien definidas.	Alta	Gerente de Proyecto
Los requerimientos no estén bien especificados.	Alta	Gerente de Producto
El cronograma maestro se modifique.	Alta	Gerente de Proyecto
Toma de decisiones a última hora.	Alta	Gerente de Proyecto
Inestabilidad de los componentes desarrollados.	Alta	Gerente de Producto
Incremento de cambios de los requerimientos no previstos.	Alta	Diseñadores, Gerente de Proyecto, Gerente de Proyecto

Terminología de la documentación

A continuación se incluyen términos generales incluidos en el documento:

Stakeholders: Son los patrocinadores del proyecto

Riesgo: Situación que provienen de cualquier suposición o que se han anticipado durante el proceso de testing.

Anexos de revisión de código. Lógica de negocio

Revisión de código de la lógica de negocio BL

Revisión de Oferta

Archivo: ActividadExtracurricular.cs

Clase: ActividadExtracurricular

Método o Propiedad	Validada	Observación
public DateTime Fecha	No	
public string Tipo	No	Campo Abierto propenso a cualquier inyección
public string Estado	No	Campo Abierto propenso a cualquier inyección
public int VersionRegistro	No	Solo debe tener el método get Publico pero también tiene el set publico
public int ServicioId	No	
public string Observación	No	Campo Abierto propenso a cualquier inyección
public int ExpedienteEstudioId	No	
public int ExpedientePlanId	No	
public int Créditos	No	
public string Diferida	No	Campo Abierto propenso a cualquier inyección

Archivo: ActividadExtracurricularAmbitos.cs

Clase: ActividadExtracurricularAmbito

Método o Propiedad	Validada	Observación
public int ModalidadID	No	
public string Modalidad	No	Campo Abierto propenso a cualquier inyección
public int TipoEstudioID	No	
public string TipoEstudio	No	Campo Abierto propenso a cualquier inyección
public int AreaAcademicaID	No	
public string AreaAcademica	No	Campo Abierto propenso a cualquier inyección
public int UnidadAcademicaID	No	

Archivo: ComponenteGrupos.cs

Clase: ComponenteGrupo

Método o Propiedad	Validada	Observación
public int GrupoId	Si	
public int CupoOfertado	No	
public int CupoReservado	No	
public int CupoConfirmado	No	Solo debe tener el método get Publico pero también tiene el set publico

Archivo: DocenteHorarios.cs

Clase: DocenteHorario

Método o Propiedad	Validada	Observación
public int OfertaDocenteId	Si	
public int HorarioId	No	
public int CentroId	Si	
public string Centro	No	Este dato no se envía a persistir en la base de datos solo se lo obtiene. Solo debe tener el método get
public DateTime? VigenteDesde	Si	
public DateTime? VigenteHasta	Si	
public int HorarioVersionRegistro	No	Solo debe tener el método get Publico pero también tiene el set publico
public int VersionRegistro	No	No se utiliza la propiedad

Archivo: GrupoDocentes.cs

Clase: GrupoDocente

Método o Propiedad	Validada	Observación
public int OfertaGrupoId	No	
public int DocenteId	Si	
public DateTime FechaInicio	No	
public DateTime? FechaFin	No	Permite valores nulos
public string Estado	No	Campo Abierto propenso a cualquier inyección
public string Docente	No	Este dato no se envía a persistir en la base de datos solo se lo obtiene. Solo debe tener el método get
public string Tipo	No	Este dato no se envía a persistir en la base de datos solo se lo obtiene. Solo debe tener el método get
public int VersionRegistro	No	Solo debe tener el método get Publico pero también tiene el set publico

Archivo: Horario.cs

Clase: Horario

Método o Propiedad	Validada	Observación
public DateTime? VigenteDesde	No	Permite valores nulos
public DateTime? VigenteHasta	No	Permite valores nulos
public int VersionRegistro	No	Solo debe tener el método get Publico pero también tiene el set publico

Archivo: HorarioDetalles.cs

Clase: HorarioDetalle

Método o Propiedad	Validada	Observación
public int HorarioId	No	Solo debe tener el método get Publico pero también tiene el set publico, es un identificador
public int DiaId	Si	
public int AulaId	Si	
public DateTime? HoraDesde	Si	El tipo de dato permite valores nulos pero en la validación no permite valores nulos
public DateTime? HoraHasta	Si	El tipo de dato permite valores nulos pero en la validación no permite valores nulos
public int VersionRegistro	No	Solo debe tener el método get Publico pero también tiene el set publico, Valor para control de concurrencia
public int EdificioId	Si	
public string Aula	No	Valor que NO se persiste en la BD, solo es recuperado pero tiene el metodo set publico
public string Dia	No	Valor que NO se persiste en la BD, solo es recuperado pero tiene el metodo set publico
public string Edificio	No	Valor que NO se persiste en la BD, solo es recuperado pero tiene el metodo set publico

Archivo: Oferta.cs

Clase: Oferta

Método o Propiedad	Validada	Observación
public int EstadoId	Si	
public int UnidadOperativaId	Si	
public int CalendarioId	Si	
public int ProcesoAcademicoId	Si	
public DateTime? VigenteDesde	No	
public DateTime? VigenteHasta	No	
public string Comentario	Si	
public int? TipoRestriccionCentroId	No	Permite tener valores nulos
public int VersionRegistro	No	
public int PlanEstudioId	No	Valor que no se persiste

Archivo: OfertaCentros.cs

Clase: OfertaCentro

Método o Propiedad	Validada	Observación
public int CentroId	Si	
public string NombreCentro	No	Valor que NO se persiste en la BD, solo es recuperado pero tiene el metodo set publico

Archivo: OfertaHorarios.cs

Clase: OfertaHorario

Método o Propiedad	Validada	Observación
public string FechaVigencia	No	Maneja un tipo de dato <SmartDate> que para exponerlo como propiedad es un string. Es un tipo de dato especial de CSLA para el manejo de fechas. Pero no se lo recomienda para aplicaciones web
public string FechaFinalizacion	No	Maneja un tipo de dato <SmartDate> que para exponerlo como propiedad es un string. Es un tipo de dato especial de CSLA para el manejo de fechas. Pero no se lo recomienda para aplicaciones web

Archivo: OfertaPlanEstudios.cs

Clase: OfertaPlanEstudio

Método o Propiedad	Validada	Observación
public int OfertaAcademicaId	No	
public int PlanConfiguracionId	No	

Archivo: OfertaProcesoAcademicoCriteria.cs

Clase: OfertaProcesoAcademicoCriteria

Método o Propiedad	Validada	Observación
public int IdTipoEstudio	Si	
public int IdModalidadEstudio	Si	
public int IdUnidadOperativa	Si	

Anexo: Revisión de código de Estructura Curricular y Programa Académico

Archivo: ComponenteEducativoIntegrados.cs
Clase: ComponenteEducativoIntegrado

Método o Propiedad	Validada	Observación
ConfiguracionComponenteEducativo	No	valor entero, solo puede contener números
ConfiguracionEstructuraCurricularDetalleId	no	valor entero, solo puede contener números
UnidadesAprobacion	Si	valor entero, solo puede contener números

Archivo: ComponenteEducativoRequisitos.cs
Clase: ComponenteEducativoRequisito

Método o Propiedad	Validada	Observación
ComponenteEducativoId	No	valor entero, solo puede contener números
Nombre	No	Cadena de texto, posible inyección de xss, no se puede inyectar SQL porque utiliza un sp estático

Archivo: EstructuraCurricular.cs
Clase: EstructuraCurricular

Método o Propiedad	Validada	Observación
Todo	Si	Clase validada correctamente

Archivo: EstructuraCurricularCodigoCmd.cs
Clase: EstructuraCurricularCodigoCmd

Método o Propiedad	Validada	Observación
Execute(string parámetro)	No	Clase comando esta solo tiene un método publico, la cadena de texto no representa ningún peligro porque sirve para ejecutar un select de un sp estático

Archivo: EstructuraCurricularVariaciones.cs
Clase: EstructuraCurricularVariacione

Método o Propiedad	Validada	Observación
DescripcionAutoriza	Si	Cadena de texto, Validación incompleta, no valida caracteres extraños, posible inyección XSS
NombreAutoriza	Si	Cadena de texto, Validación incompleta, no valida caracteres extraños, posible inyección XSS
FechaCaducidad	Si	Contiene una validación personalizada
FechaCreacion	Si	
Nombre	Si	Cadena de texto, Validación incompleta, no valida caracteres extraños, posible inyección XSS

Archivo: EstructuraCurricularVariacionNombreCmd.cs
Clase: EstructuraCurricularVariacionNombreCmd

Método o Propiedad	Validada	Observación
Execute(string parámetro)	No	Clase comando, un método publico, la cadena de texto no representa ningún peligro porque sirve para ejecutar un select de un sp estático.

Archivo: VariacionActividadesAcademicas.cs
Clase: VariacionActividadAcademica

Método o Propiedad	Validada	Observación
ComponenteEducativoId	No	valor entero, solo puede contener números
Nombre	No	Cadena de texto, Validación incompleta, no valida caracteres extraños, posible inyección XSS

Archivo: VariacionComponentesEducativos.cs
Clase: VariacionComponentesEducativos

Método o Propiedad	Validada	Observación
Todo	Si	Clase con todas las validaciones, solo tiene valores numéricos

Archivo: EstructuraCurricularCriterio.cs
Clase: EstructuraCurricularCriterio

Método o Propiedad	Validada	Observación
Todo	Si	Clase criterio, tiene como propiedades que sirven como parámetros de búsqueda, tienes un parámetro cadena de texto que se encuentra validado, y el resto de valores son numéricos, utiliza un SP estático

Archivo: ProcesoAcademicoCriterioPag.cs
Clase: ProcesoAcademicoCriterioPag

Método o Propiedad	Validada	Observación
UnidadAcademicaId	No	Clase criterio, ninguna propiedad validada, tiene como propiedades que sirven como parámetros de búsqueda, tiene un parámetro cadena de texto, y el resto de valores son numéricos, utiliza un SP estático
NivelAcademicoId	No	
ModalidadId	No	
Codigo	No	Propiedad tipo string
Nombre	No	Propiedad tipo string

Archivo: Estudio.cs
Clase: Estudio

Método o Propiedad	Validada	Observación
Todo	Si	Clase con todas la validaciones correctas

Archivo: EstudioCentros.cs
Clase: EstudioCentro

Método o Propiedad	Validada	Observación
CentroNombre	No	Cadena de texto, Validación incompleta, no valida caracteres extraños.
EstudioId	Si	Numérico
CentroId	Si	Numérico

Archivo: EstudioCompetencias.cs
Clase: EstudioCompetencias

Método o Propiedad	Validada	Observación
Todo	Si	Clase con todas la validaciones correctas

Archivo: EstudioModalidades.cs
Clase: EstudioModalidad

Método o Propiedad	Validada	Observación
PlanEstudioId	No	
EstudioModalidadId	No	
ResolucionOficial	No	
FechaAprobacion	No	

Archivo: EstudioPerfilProfesionales.cs
Clase: EstudioPerfilProfesional

Método o Propiedad	Validada	Observación
Todo	Si	Clase con todas la validaciones correctas

Archivo: EstudioRequisitoPromociones.cs
Clase: EstudioRequisitoPromocion

Método o Propiedad	Validada	Observación
Todo	Si	Clase con todas la validaciones correctas

Anexo: Revisión de código de Componentes Educativos

Archivo: ActividadExtracurricularCriterioPag
Clase: ActividadExtracurricularCriterioPag

Método o Propiedad	Validada	Observación
Todo	Si	Toda la clase esta completamente validada

Archivo: Actividades.cs
Clase: Actividad

Método o Propiedad	Validada	Observación
UnidadOperativaPropietariaId	No	Esta clase no posee ninguna validación
TipoUnidadEnseñanzaId	No	
Codigo	No	Propiedad tipo string
Nombre	No	Propiedad tipo string
Descripcion	No	Propiedad tipo string
FechaCreacion	No	
EstadoId	No	
ClasificacionID	No	
TipoActividadId	No	
AplicaParticular	No	Propiedad tipo string
Creditos	No	

Archivo: ActividadExtracurricular.cs
Clase: ActividadExtracurricular

Método o Propiedad	Validada	Observación
Todo	Si	Completamente validada

Archivo: ComponenteEducativo.cs
Clase: ComponenteEducativo

Método o Propiedad	Validada	Observación
Todo	Si	Completamente validada

Archivo: ComponenteEducativoCriteria.cs
Clase: ComponenteEducativoCriteria

Método o Propiedad	Validada	Observación
Todo	Si	Completamente validada

Archivo: ExisteCodigoComponenteEducativo.cs
Clase: ExisteCodigoComponenteEducativo

Método o Propiedad	Validada	Observación
Execute(string codigo)	No	Clase comando que recibe un parámetro string pero este parámetro no esta siendo validado

Archivo: ComponenteEducativoConfiguracionCriteria.cs
Clase: ComponenteEducativoConfiguracionCriteria

Método o Propiedad	Validada	Observación
Todo	Si	Clase completamente validada

Archivo: ComponenteUnidadEnseñanza.cs
Clase: ComponenteUnidadEnseñanza

Método o Propiedad	Validada	Observación
TipoId	No	
TipoNombre	No	Parametro tipo String
VersionId	No	
Nombre	No	Parametro tipo String
Descripcion	No	Parametro tipo String

Archivo: ComponenteUnidadEnseñanza.cs
Clase: ComponenteUnidadEnseñanza

Método o Propiedad	Validada	Observación
Todo	Si	Clase completamente validada

Archivo: ConfiguracionComponenteEducativoVersion.cs
Clase: ConfiguracionComponenteEducativoVersion

Método o Propiedad	Validada	Observación
Todo	Si	Clase completamente validada

Archivo: ConfiguracionCriteria.cs
Clase: ConfiguracionCriteria

Método o Propiedad	Validada	Observación
Todo	Si	Clase completamente validada

Archivo: ConfiguracionCriteria.cs
Clase: ConfiguracionCriteria

Método o Propiedad	Validada	Observación
Todo	Si	En esta clase solo una propiedad no tiene validación todas las demás propiedades están debidamente validadas, esta propiedad es del tipo string por lo que representa un riesgo de inyección, esta propiedad no se le esta seteando ningún valor dentro de la capa superior, solo en la lógica de acceso a datos

Archivo: ConfiguracionUnidadEnseñanza.cs
Clase: ConfiguracionUnidadEnseñanza

Método o Propiedad	Validada	Observación
Todo	Si	Clase completamente validada

Archivo: ComponenteEducativoConfiguracionCriteria.cs
Clase: ComponenteEducativoConfiguracionCriteria

Método o Propiedad	Validada	Observación
Todo	Si	Clase completamente validada

Archivo: ConfiguracionVersionCriterio.cs
Clase: ConfiguracionVersionCriterio

Método o Propiedad	Validada	Observación
VersionId	No	Esta clase no tiene ninguna validación, tiene dos propiedades una del tipo int y otro date time por lo que no presenta ningún problema de seguridad
FechaDesde	Si	

Archivo: FormaImpartirCriterio.cs
Clase: FormaImpartirCriterio

Método o Propiedad	Validada	Observación
	No	Esta clase no tiene ninguna validación, tiene varias propiedades todas del tipo int por lo que no presenta ningún problema de seg.

Archivo: TipoEstudioCriterio.cs
Clase: TipoEstudioCriterio

Método o Propiedad	Validada	Observación
AsignaturaId	No	Esta clase no tiene ninguna validación, tiene varias propiedades todas del tipo int por lo que no presenta ningún problema de seguridad
UnidadPropietariaAdministradaId	No	

Archivo: UnidadEnseñanzaConfiguracionCriterio.cs
Clase: UnidadEnseñanzaConfiguracionCriterio

Método o Propiedad	Validada	Observación
UnidadOperativaId	No	
Codigo	No	Propiedad tipo string
Nombre	No	Propiedad tipo string

Archivo: Actividad.cs
Clase: Actividad

Método o Propiedad	Validada	Observación
Todo	Si	Clase completamente validada

Archivo: Asignatura.cs
Clase: Asignatura

Método o Propiedad	Validada	Observación
Todo	Si	Clase completamente validada

Archivo: AsignaturaBasicaCriteria.cs
Clase: AsignaturaBasicaCriteria

Método o Propiedad	Validada	Observación
Todo	Si	Clase completamente validada

Archivo: AsignaturaCriteria.cs
Clase: AsignaturaCriteria

Método o Propiedad	Validada	Observación
Todo	Si	Clase completamente validada

Archivo: AsignaturaCriteriaPag.cs
Clase: AsignaturaCriteriaPag

Método o Propiedad	Validada	Observación
Todo	Si	Clase completamente validada

Archivo: UnidadEnseñanzaMallaCriteria.cs
Clase: UnidadEnseñanzaMallaCriteria

Método o Propiedad	Validada	Observación
Todo	Si	Clase completamente validada

Archivo: UnidadEnseñanzaAdministradora.cs
Clase: UnidadEnseñanzaAdministradora

Método o Propiedad	Validada	Observación
Todo	No	Clase obsoleta

Archivo: UnidadEnseñanzaCompetencia.cs
Clase: UnidadEnseñanzaCompetencia

Método o Propiedad	Validada	Observación
Todo	No	Clase obsoleta

Archivo: AsignaturaCompetenciaCriteria.cs
Clase: AsignaturaCompetenciaCriteria

Método o Propiedad	Validada	Observación
Todo	No	Clase obsoleta

Anexo: Revisión de código del Modulo de Matrícula

Archivo: GestorHorarios.cs
Clase: GestorHorarios

Método o Propiedad	Validada	Observación
Todos	No	La clase no necesita validaciones por que esta clase contiene algoritmos de validacion de negocio, sus datos no son persistibles

Archivo: HorarioCriteria.cs
Clase: HorarioCriteria

Método o Propiedad	Validada	Observación
OfertaAcademicaDetalleId	No	
VersionConfiguracionCOEId	No	
ParaleloOfertadoId	No	

Archivo: Horarios.cs
Clase: Horario

Método o Propiedad	Validada	Observación
VigenteDesde	No	Tipo datetime no representa ningún problema de seguridad
VigenteHasta	No	Tipo datetime no representa ningún problema de seguridad
Dia	No	Tipo datetime no representa ningún problema de seguridad
HoraInicio	No	Tipo datetime no representa ningún problema de seguridad
HoraFin	No	Tipo datetime no representa ningún problema de seguridad

Archivo: EstudianteBecaCriterio.cs
Clase: EstudianteBecaCriterio

Método o Propiedad	Validada	Observación
EstudianteId	No	Tipo Int
ExpedienteEstudioId	No	Tipo Int
Fecha	No	Tipo datetime

Archivo: MatriculaBecas.cs
Clase: MatriculaBeca

Método o Propiedad	Validada	Observación
BecaEstudianteId	No	int
Nombre	No	Propiedad tipo string por donde se puede hacer una inyección
Acumulativa	No	bool
Tipo	No	int
Tipo_Valor	No	Propiedad tipo string por donde se puede hacer una inyección
Valor	No	float
Monto	No	double

Archivo: MatriculaCuotas.cs
Clase: MatriculaCuota

Método o Propiedad	Validada	Observación
Monto	No	double
FechaEmision	No	Propiedad tipo string por donde se puede hacer una inyección
FechaPago	No	Propiedad tipo string por donde se puede hacer una inyección

Archivo: ComponenteGrupo.cs
Clase: ComponenteGrupo

Método o Propiedad	Validada	Observación
GrupoId	Si	
CupoOfertado	Si	
CupoReservado	No	
CupoConfirmado	No	

Archivo: CruceHorario.cs
Clase: CruceHorario

Método o Propiedad	Validada	Observación
Todas	No	Clase DTO utilizada para el paso de mensajes entre objetos

Archivo: GrupoDocente.cs
Clase: GrupoDocente

Método o Propiedad	Validada	Observación
OfertaGrupoId	No	int
DocenteId	No	int
FechaInicio	No	dateTime
FechaFin	No	dateTime
Estado	No	string, peligro de inyección
Tipo	No	string, peligro de inyección

Archivo: GrupoHorario.cs
Clase: GrupoHorario

Método o Propiedad	Validada	Observación
VigenteDesde	No	dateTime
VigenteHasta	No	dateTime

Archivo: HorarioJornadas.cs
Clase: HorarioJornada

Método o Propiedad	Validada	Observación
Dia	No	string, peligro de inyección
Inicio	No	dateTime
Fin	No	dateTime

Archivo: ItemParalelo.cs
Clase: ItemParalelo

Método o Propiedad	Validada	Observación
Nombre	No	string, peligro de inyección
CuposDisponibles	No	int

Archivo: ParaleloComponentes.cs
Clase: ParaleloComponente

Método o Propiedad	Validada	Observación
ComponenteID	No	int
ParaleloID	No	int

Archivo: SolicitudItems.cs
Clase: SolicitudItem

Método o Propiedad	Validada	Observación
Nombre	No	string, peligro de inyección
Codigo	No	string, peligro de inyección
Nivel	No	int
Costo	No	double
Creditos	No	int
GrupoDeCreditos	No	string, peligro de inyección
Repeticiones	No	int
Aprobaciones	No	int
Observacion	No	string, peligro de inyección
Restricciones	No	string, peligro de inyección

Archivo: OfertaUnidadEnsenanzaCalendario.cs
Clase: OfertaUnidadEnsenanzaCalendarioOfertaUnidadEnsenanzaCalendario

Método o Propiedad	Validada	Observación
Todas	No	La clase contiene propiedades del tipo int lo que no representa problemas de seguridad

Archivo: PotencialCriterio.cs
Clase: PotencialCriterio

Método o Propiedad	Validada	Observación
PlanEstudioId	No	int
EstudianteId	No	int
MatriculaId	No	int
CicloAcademicoId	No	int
CentroEstudioId	No	int
TipocentroEstudioId	No	int
FormaPagoId	No	int
TipoEstudioId	No	int
ConvenioId	No	int
TipomatriculaId	No	int
EstudioId	No	int
ModalidadId	No	int
TipoAprobacionId	No	int
TipoPagoId	No	int
TieneConvenio	No	bool
TieneDescuentoAsignaturas	No	bool
TieneTasaAdministrativa	No	bool

Archivo: CuposDisponiblesCmd.cs
Clase: CuposDisponiblesCmd

Método o Propiedad	Validada	Observación
int Execute(int IdOfertaDet)	No	Método execute recibe un parámetro tipo int que no es validado

Archivo: Matricula.cs
Clase: Matricula

Método o Propiedad	Validada	Observación
Todas	Si	Clase correcta

Archivo: MatriculaDetalles.cs
Clase: MatriculaDetalle

Método o Propiedad	Validada	Observación
Todas	Si	Clase correcta

Archivo: MatriculaDTO.cs
Clase: MatriculaDTO

Método o Propiedad	Validada	Observación
Todas	No	Clase DTO, utilizada para el paso de información entre objetos

Archivo: MatriculaSolicitudes.cs
Clase: MatriculaSolicitud

Método o Propiedad	Validada	Observación
Fecha	No	Date
Tipo	No	string, peligro de inyección
Estado	No	string, peligro de inyección
ServicioId	No	int
Observacion	No	string, peligro de inyección
ExpedienteEstudioId	No	int
ExpedientePlanId	No	int
Creditos	No	int
Diferida	No	string, peligro de inyección

Archivo: SolicitudCriteria.cs
Clase: SolicitudCriteria

Método o Propiedad	Validada	Observación
EstudianteId	No	int
ExpedienteEstudioId	No	int
ExpedientePlanId	No	int
CicloAcademicoId	No	int
Anio	No	int

Anexo: Revisión de código. Interfaz gráfica de usuario UI

Revisión de Estructura Curricular

Archivo: BuscadorEstructuraCurricular.aspx
Clase: BuscadorEstructuraCurricular

Método o Propiedad	Validada	Observación
Page_Load	No	Método no contiene manejo de excepciones
btnBuscarClick	No	Método no contiene manejo de excepciones
gplEstructuraCurricular_Command	No	Método no contiene manejo de excepciones

Archivo: ucActividadesAcademicas.aspx
Clase: ucActividadesAcademicas

Método o Propiedad	Validada	Observación
Page_Load	No	Método no contiene manejo de excepciones

Archivo: ucBuscadorActividadesAcademicas.aspx
Clase: ucBuscadorActividadesAcademicas

Método o Propiedad	Validada	Observación
Page_Load	No	Método no contiene manejo de excepciones

Archivo: ucBuscadorComponentesEducativosPorCreditos.ascx
Clase: ucBuscadorComponentesEducativosPorCreditos

Método o Propiedad	Validada	Observación
Page_Load	No	Método no contiene manejo de excepciones

Archivo: ucConfiguracionComponenteEducativoPorComponentes.ascx
Clase: ucConfiguracionComponenteEducativoPorComponentes

Método o Propiedad	Validada	Observación
Page_Load	No	Método no contiene manejo de excepciones

Archivo: ucConfiguracionComponenteEducativoPorCreditos.ascx
Clase: ucConfiguracionComponenteEducativoPorCreditos

Método o Propiedad	Validada	Observación
Page_Load	No	Método no contiene manejo de excepciones

Archivo: ucConfigurarEstructuraCurricular.ascx
Clase: ucConfigurarEstructuraCurricular

Método o Propiedad	Validada	Observación
Page_Load	No	Método no contiene manejo de excepciones
CrearNuevaAsignatura	No	Método no contiene manejo de excepciones
btnCancelarPlanEstudio_Click	No	Método no contiene manejo de excepciones
btnAceptarConfiguracion_Click	No	Método no contiene manejo de excepciones

Archivo: ucDistribucionCreditos.ascx
Clase: ucDistribucionCreditos

Método o Propiedad	Validada	Observación
Page_Load	No	Método no contiene manejo de excepciones

Archivo: ucEstructuraPorComponentes.ascx
Clase: ucEstructuraPorComponentes

Método o Propiedad	Validada	Observación
Page_Load	No	Método no contiene manejo de excepciones
btnGuardar_clik	No	Método no contiene manejo de excepciones

Archivo: ucEstructuraPorComponentes.ascx
Clase: ucEstructuraPorComponentes

Método o Propiedad	Validada	Observación
Page_Load	No	Método no contiene manejo de excepciones
btnGuardar_clik	No	Método no contiene manejo de excepciones

Archivo: ucEstructuraPorCreditos.ascx
Clase: ucEstructuraPorCreditos

Método o Propiedad	Validada	Observación
btnGuardar_clik	No	Método no contiene manejo de excepciones

Archivo: ucEstructuraPorCreditos.ascx
Clase: ucEstructuraPorCreditos

Método o Propiedad	Validada	Observación
btnGuardar_clik	No	Método no contiene manejo de excepciones

Archivo: ucEstudioEstructuraCurricular
Clase: ucEstudioEstructuraCurricular

Método o Propiedad	Validada	Observación
stoPlanesEstudios_Refresh	No	Método no contiene manejo de excepciones

Archivo: ucListarEstructuraCurricular
Clase: ucListarEstructuraCurricular

Método o Propiedad	Validada	Observación
Page_Load	No	Método no contiene manejo de excepciones

Archivo: ucVersionesConfiguracion
Clase: ucVersionesConfiguracion

Método o Propiedad	Validada	Observación
cbbVersion_Select	No	Método no contiene manejo de excepciones

Archivo: wccBuscadorAvanzadoAsignaturas.cs
Clase: wccBuscadorAvanzadoAsignaturas

Método o Propiedad	Validada	Observación
Asignaturas_RefreshData	No	No hay manejo de excepciones. UserControl personalizado.

Archivo: wccBuscadorProgramaAcademico.cs
Clase: wccBuscadorProgramaAcademico

Método o Propiedad	Validada	Observación
ProgramasAcademicos_RefreshData	No	No hay manejo de excepciones. UserControl personalizado.

Archivo: wccEstructuraCurricularDatosGenerales
Clase: wccEstructuraCurricularDatosGenerales

Método o Propiedad	Validada	Observación
ProgramasAcademicos_RefreshData	No	No hay manejo de excepciones. UserControl personalizado.

Archivo: wccBuscadorProgramaAcademico
Clase: wccBuscadorProgramaAcademico

Método o Propiedad	Validada	Observación
ProgramasAcademicos_RefreshData	No	No hay manejo de excepciones. UserControl personalizado.

Archivo: wccEstructuraCurricularDatosGenerales
Clase: wccEstructuraCurricularDatosGenerales

Método o Propiedad	Validada	Observación
ProgramasAcademicos_RefreshData	Si	User control personalizado

Archivo: wccInformaciondeVariaciones
Clase: wccInformaciondeVariaciones

Método o Propiedad	Validada	Observación
ProgramasAcademicos_RefreshData	Si	User control personalizado

Anexo: Revisión de código de Oferta Académica

Archivo: BuscadorGrupoCreditos.aspx
Clase: BuscadorGrupoCreditos

Método o Propiedad	Validada	Observación
Page_Load	No	No se tiene un manejo de excepciones

Archivo: ucDistribucionGrupoCredito.ascx
Clase: ucDistribucionGrupoCredito.ascx

Método o Propiedad	Validada	Observación
Page_Load	No	No se tiene un manejo de excepciones

Archivo: ProcesoAcademicoList.aspx
Clase: ProcesoAcademicoList

Método o Propiedad	Validada	Observación
Page_Load	No	No se tiene un manejo de excepciones

Archivo: ucConfigurarCondicion.ascx
Clase: ucConfigurarCondicion

Método o Propiedad	Validada	Observación
Page_Load	No	No se tiene un manejo de excepciones

Archivo: BusquedaCicloAcademico.aspx
Clase: BusquedaCicloAcademico

Método o Propiedad	Validada	Observación
Page_Load	No	No se tiene un manejo de excepciones

Archivo: ucRegistroCicloAcademico.ascx
Clase: ucRegistroCicloAcademico

Método o Propiedad	Validada	Observación
Page_Load	No	No se tiene un manejo de excepciones

Archivo: EstudioWeb.aspx
Clase: EstudioWeb

Método o Propiedad	Validada	Observación
Page_Load	No	No se tiene un manejo de excepciones
tabDatosGenerales_Render	No	No se tiene un manejo de excepciones
stoPlanesEstudios_Refresh	No	No se tiene un manejo de excepciones
btnNuevoPlanEstudio_Click	No	No se tiene un manejo de excepciones
gdpPlanesEstudioCommandColumn_Click	No	No se tiene un manejo de excepciones
btnCancelar_Click	No	No se tiene un manejo de excepciones

Archivo: ucBusquedaCentrosUniversitarios.ascx
Clase: ucBusquedaCentrosUniversitarios

Método o Propiedad	Validada	Observación
Page_Load	No	No se tiene un manejo de excepciones

Archivo: ucResponsableEstudio.ascx
Clase: ucResponsableEstudio

Método o Propiedad	Validada	Observación
Page_Load	No	No se tiene un manejo de excepciones

Archivo: BuscadorEstudios.aspx
Clase: BuscadorEstudios

Método o Propiedad	Validada	Observación
Page_Load	No	No se tiene un manejo de excepciones

Archivo: GrupoComun.aspx
Clase: GrupoComun

Método o Propiedad	Validada	Observación
Todo	Si	Correcto manejo de excepciones

Archivo: ucAdministrarGrupoCreditos.ascx
Clase: ucAdministrarGrupoCreditos

Método o Propiedad	Validada	Observación
Todo	Si	Correcto manejo de excepciones

Archivo: ucBusquedaEmpleados.aspx
Clase: ucBusquedaEmpleados

Método o Propiedad	Validada	Observación
Todo	Si	Correcto manejo de excepciones

Archivo: usBusquedaDocumento.ascx
Clase: usBusquedaDocumento

Método o Propiedad	Validada	Observación
Todo	Si	Correcto manejo de excepciones

Archivo: ucBusquedaUnidadPropietaria.ascx
Clase: ucBusquedaUnidadPropietaria

Método o Propiedad	Validada	Observación
Todo	Si	Correcto manejo de excepciones

Archivo: ucBusquedaAsignatura.ascx
Clase: ucBusquedaAsignatura

Método o Propiedad	Validada	Observación
Todo	Si	Correcto manejo de excepciones

Archivo: ucConfigurarActividad.ascx
Clase: ucConfigurarActividad

Método o Propiedad	Validada	Observación
Todo	Si	Correcto manejo de excepciones

Archivo: ucConfigurarActividades.ascx
Clase: ucConfigurarActividades

Método o Propiedad	Validada	Observación
Todo	Si	Correcto manejo de excepciones

Archivo: ucConfigurarProcesoAcademico.ascx
Clase: .ascx

Método o Propiedad	Validada	Observación
Todo	Si	Correcto manejo de excepciones

Archivo: ucAdministrarEstudio.ascx
Clase: ucAdministrarEstudio

Método o Propiedad	Validada	Observación
Todo	Si	Correcto manejo de excepciones

Anexo: Revisión de código de Programa Académico

Archivo: EstudioWeb.aspx
Clase: EstudioWeb.cs

Método o Propiedad	Validada	Observación
Page_Load	No	No se tiene un manejo de excepciones
tabDatosGenerales_Render	No	
stoPlanesEstudios_Refresh	No	Potencial pagina en riesgo no se tiene ningun manejo de excepcionesen toda la pagina
btnNuevoPlanEstudio_Click	No	
gdpPlanesEstudioCommandColumn_Click	No	
btnCancelar_Click	No	

Archivo: ucAdministrarEstudio.ascx
Clase: ucAdministrarEstudio

Método o Propiedad	Validada	Observación
Page_Load	No	No se tiene un manejo de excepciones

Archivo: ucBusquedaCentrosUniversitarios.ascx
Clase: ucBusquedaCentrosUniversitarios

Método o Propiedad	Validada	Observación
Page_Load	No	No se tiene un manejo de excepciones

Archivo: ucResponsableEstudio.ascx
Clase: ucResponsableEstudio

Método o Propiedad	Validada	Observación
Page_Load	No	No se tiene un manejo de excepciones

Archivo: GrupoComun.aspx
Clase: GrupoComun

Método o Propiedad	Validada	Observación
Todo	Si	Manejo de excepciones exitoso

Archivo: ucAdministrarGrupoCreditos.ascx
Clase: ucAdministrarGrupoCreditos

Método o Propiedad	Validada	Observación
Todo	Si	Manejo de excepciones exitoso

Archivo: ucBusquedaEmpleados.aspx
Clase: ucBusquedaEmpleados

Método o Propiedad	Validada	Observación
Todo	Si	Manejo de excepciones exitoso

Archivo: ucBusquedaUnidadPropietaria.ascx
Clase: ucBusquedaUnidadPropietaria

Método o Propiedad	Validada	Observación
Todo	Si	Manejo de excepciones exitoso

Archivo: ucBusquedaAsignatura.ascx
Clase: ucBusquedaAsignatura

Método o Propiedad	Validada	Observación
Todo	Si	Manejo de excepciones exitoso

Archivo: ucBusquedaActividadExtracurricular.ascx
Clase: ucBusquedaActividadExtracurricular

Método o Propiedad	Validada	Observación
Todo	Si	Manejo de excepciones exitoso

Archivo: ucBusquedaActividadExtracurricular.ascx
Clase: ucBusquedaActividadExtracurricular

Método o Propiedad	Validada	Observación
Todo	Si	Manejo de excepciones exitoso

Archivo: ucDistribucionGrupoCredito.ascx
Clase: ucDistribucionGrupoCredito

Método o Propiedad	Validada	Observación
Page_Load	No	No se tiene un manejo de excepciones

Archivo: BuscadorGrupoCreditos.aspx
Clase: BuscadorGrupoCreditos

Método o Propiedad	Validada	Observación
Page_Load	No	No se tiene un manejo de excepciones

Archivo: ucVersionesAsignatura.ascx
Clase: ucVersionesAsignatura

Método o Propiedad	Validada	Observación
Page_Load	No	No se tiene un manejo de excepciones

Archivo: ucPlanEstudioModalidades.ascx
Clase: ucPlanEstudioModalidades

Método o Propiedad	Validada	Observación
Page_Load	No	No se tiene un manejo de excepciones

Archivo: BuscadorEstudios.aspx
Clase: BuscadorEstudios

Método o Propiedad	Validada	Observación
Page_Load	No	No se tiene un manejo de excepciones

Archivo: ucConfigurarAsignatura
Clase: ucConfigurarAsignatura.ascx

Método o Propiedad	Validada	Observación
Page_Load	No	No se tiene un manejo de excepciones

Archivo: ucConfigurarExcepcion.ascx
Clase: ucConfigurarExcepcion.ascx

Método o Propiedad	Validada	Observación
Page_Load	No	No se tiene un manejo de excepciones

Archivo: ucConfigurarRestriccion.ascx
Clase: ucConfigurarRestriccion

Método o Propiedad	Validada	Observación
Page_Load	No	No se tiene un manejo de excepciones

Anexo: Revisión de código de base de datos.

Revisión de paquetes de Base de datos: ADM (Administración)

Paquete	Procedimiento dinámico	Cant. de SP	Observaciones
PKG_ADM_CAMPO_EXTENDIDO		9	Paquete seguro
PKG_ADM_CATALOGO	PRC_OBT_ITEMS_CATALOGO_IN	22	El paquete contiene un procedimiento almacenado dinámico donde se recibe dos cadenas de texto posible inyección SQL, pero los datos son enviados como parámetros son validados
PKG_ADM_CATALOGO_EXTENDIDO	PRC_OBT_CATALOGO_EXTENDIDO	1	Paquete seguro
PKG_ADM_PARAMETRO		5	Paquete seguro
PKG_ADM_PROPIEDAD_DINAMICA	PRC_OBT_PROPIEDADES_DINAMICAS	10	Paquete seguro
PKG_ADM_REGLAS		2	Paquete seguro
PKG_ADM_SECUENCIA	PRC_OBT_SECUENCIA	1	El paquete contiene un procedimiento almacenado dinámico donde se recibe una cadenas de texto posible inyección SQL, la variable enviada es tomada de una clase de constantes,
PKG_ADM_SISTEMA		5	Paquete seguro
PKG_ADM_TABLA	PRC_OBT_COLUMNAS	12	El paquete contiene dos procedimiento almacenado dinámicos donde uno de ellos recibe un parámetro varchar, pero de donde se lo llama al paquete los datos están validados
PKG_ADM_UTILIDADES		1	Paquete seguro

Revisión de paquetes de Base de datos: SGA (Gestión Académica)

Paquete	Procedimiento	Cant. de SP	Observaciones
PKG_SGA_ACTIVIDAD	PRC_BUSCAR_ACTIVIDADES	7	Procedimiento almacenado dinámico, con un campo de entrada varchar, posible inyección, clase no validada, no se utiliza la clase que utiliza este procedimiento
PKG_SGA_ACTIVIDAD_PROCESO		3	Paquete seguro
PKG_SGA_ANULACION_ASIGNATURAS		2	Paquete seguro
PKG_SGA_ASIGNATURA		14	Paquete seguro
PKG_SGA_AULA	PRC_OBT_AULAS	7	Procedimiento almacenado dinámico, con un campo de entrada varchar, potencial riesgo de inyección, la clase incompleta validaciones, la cadena de texto que se envía como parámetro
PKG_SGA_AULA_RECURSO		4	Paquete seguro
PKG_SGA_AUTORIZACION	PRC_OBT_ITEMS_CATALOGO	9	Procedimiento almacenado dinámico, con dos campos de entrada varchar, potencial riesgo de inyección SQL, pero los datos de entrada no son ingresados por el usuario, son dos parámetros de constantes, del sistema
PKG_SGA_CALEN_ACT_COND_FECHA		6	Paquete seguro
PKG_SGA_CALENDARIO		6	Paquete seguro
PKG_SGA_CALENDARIO_ACADEMICO		5	Paquete seguro
PKG_SGA_CALENDARIO_ACT_COND		6	Paquete seguro
PKG_SGA_CALENDARIO_ACTIVIDAD		7	Paquete seguro
PKG_SGA_CANAL_PAGOS		1	Paquete seguro
PKG_SGA_CCO_DEPENDIENTE		5	Paquete seguro
PKG_SGA_CCO_DEUDOR		5	Paquete seguro
PKG_SGA_CCO_SERVICIO		5	Paquete seguro
PKG_SGA_CCO_TIPO_CREDITO		5	Paquete seguro
PKG_SGA_CCO_TIPO_PAGO		5	Paquete seguro
PKG_SGA_CENTRO_STU_EJECUCION		6	Paquete seguro
PKG_SGA_CENTRO_UNIVERSITARIO	PRC_OBT_CENTRO_UNIVER_NOADD	4	Procedimieto almacenado dinamíco, con un campo de entrada varchar, posible inyección, clase no validada, pero no se esta utilizando la clase que utiliza este procedimento
PKG_SGA_CENTROS_EVALUACIONES		15	Paquete seguro, a pesar que tiene dos paquetes dinamicos pero solo ser reciben valores numeriocs
pkg_sga_ciclo_academico		10	Paquete seguro
PKG_SGA_COMPETENCIA_ASIGNATURA		3	Paquete seguro
PKG_SGA_COMPONENTE_EDUCATIVO		11	Paquete seguro
PKG_SGA_COMPONENTE_INTEGRAD		5	Paquete seguro
PKG_SGA_CONF_DISTR_CRED		6	Paquete seguro
PKG_SGA_CONVENIO		12	Paquete seguro
PKG_SGA_CONVENIO_CONFIGURACION		5	Paquete seguro
PKG_SGA_DIRECCION		6	Paquete seguro
PRC_INS_DIRECCION_ENTE		1	Paquete seguro
PKG_SGA_DOCENTE		2	Paquete seguro
PKG_SGA_DOCUMENTO		1	Paquete seguro
PKG_SGA_ECC_DETALLE		6	Paquete seguro
PKG_SGA_ECU_CED_REQUISITO		5	Paquete seguro

Paquete	Procedimiento	Cant. de SP	Observaciones
PKG_SGA_ECU_CONFIGURACION		6	Paquete seguro
PKG_SGA_EDIFICIO		5	Paquete seguro
PKG_SGA_EMPLEADO		2	Paquete seguro
PKG_SGA_EMPRESA		5	Paquete seguro
PKG_SGA_ENTE		4	Paquete seguro
PKG_SGA_ESTRUCTURA_CURRICULAR		7	Paquete seguro
PKG_SGA_ESTUDIANTE		5	Paquete seguro
PKG_SGA_ESTUDIANTE_DEUDA		1	Paquete seguro
PKG_SGA_ESTUDIANTE_IMPEDIMENT		5	Paquete seguro
PKG_SGA_ESTUDIO		5	Paquete seguro, este paquete tiene un procedimineto dinamico, pero solo recibe un parametros numericos lo que no implica problemas
PKG_SGA_ESTUDIO_MODALIDAD		7	Paquete seguro
PKG_SGA_ESTUDIO_REQUISITO		5	Paquete seguro
PKG_SGA_ESTUDIOS_REALIZADOS		5	Paquete seguro
PKG_SGA_EVALUACIONES		1	Paquete seguro
PKG_SGA_EXPEDIENTE		13	Paquete seguro
PKG_SGA_EXPEDIENTE_ESTUDIO		7	Paquete seguro
PKG_SGA_EXPEDIENTE_ESTUDIO_UEN		4	Paquete seguro
PKG_SGA_CENTRO_UNIVERSITARIO	PRC_OBT_CENTRO_UNIVER_NOADD	4	Este paquete contiene un procedimiento dinamico que recibe un parametro varchar, y esta no tiene ninguna validacion en la aplicación
PKG_SGA_EXPEDIENTE_PLAN		10	Paquete seguro
PKG_SGA_GCR_TES		1	Paquete seguro
pkg_sga_grupo_credito		4	Paquete seguro
PKG_SGA_HORARIO		4	Paquete seguro
PKG_SGA_HORARIO_DETALLE		6	Paquete seguro
PKG_SGA_IDENTIFICACION		5	Paquete seguro
PKG_SGA_IMPEDIMENTO		14	Paquete seguro
PKG_SGA_IMPEDIMENTO_MOTIVO		8	Paquete seguro
PKG_SGA_MATRICULA		7	Paquete seguro
PKG_SGA_MODALIDAD		3	Paquete seguro
pkg_sga_nivel_academico		1	Paquete seguro
pkg_sga_oferta		7	Paquete seguro
pkg_sga_oferta_componente		7	Paquete seguro
pkg_sga_oferta_docente		6	Paquete seguro
pkg_sga_oferta_grupo		7	Paquete seguro
PKG_SGA_OFERTA_HORARIO		6	Paquete seguro
pkg_sga_oferta_paralelo		7	Paquete seguro
pkg_sga_oferta_plan_estudio		7	Paquete seguro
pkg_sga_oferta_restriccion_cen		5	Paquete seguro
PKG_SGA_OFERTA_UNIDAD_ENSENANZ		6	Paquete seguro
PKG_SGA_ORGANIZACION		10	Paquete seguro
pkg_sga_oue_cupo_stu		6	Paquete seguro
PKG_SGA_PARAMETRO_VALIDADOR		1	Paquete seguro
PKG_SGA_PERSONA		8	Paquete seguro
PKG_SGA_PERSONA_ROL		4	Paquete seguro
PKG_SGA_PLAN_CFG_ACTIVIDADES_ACADEMICAS		5	Paquete seguro
PKG_SGA_PLANTILLA		11	Paquete seguro
PKG_SGA_PLANTILLA_ACTIVIDAD		4	Paquete seguro

Paquete	Procedimiento	Cant. de SP	Observaciones
PKG_SGA_PROCESO_ACADEMICO		7	Paquete seguro
PKG_SGA_PROGRAMA_ACADEMICO	PRC_OBT_PROGRAMA_ACADEMICOS	5	Este paquete contiene un procedimiento dinamico que recibe dos parametro varchar, y esta no tiene ninguna validacion en la aplicación
PKG_SGA_REGLA		1	Paquete seguro
PKG_SGA_REPOSITORIO_COBROS		3	Paquete seguro
PKG_SGA_REQUI_PROGRA_ACADEMICO		6	Paquete seguro
PKG_SGA_REQUISITO		5	Paquete seguro
PKG_SGA_REQUISITO_AMBITO		7	Paquete seguro
PKG_SGA_REQUISITO_ASIGNATURA		6	Paquete seguro
PKG_SGA_REQUISITO_DOCUMENTO		6	Paquete seguro
PKG_SGA_REQUISITO_PAR_VALID		5	Paquete seguro
PKG_SGA_REQUISITO_TITULO		6	Paquete seguro
PKG_SGA_RES_COM_EDUCATIVO		7	Paquete seguro
PKG_SGA_RESOLUCION		4	Paquete seguro
PKG_SGA_RESOLUCION_MODALIDAD		4	Paquete seguro
PKG_SGA_RESPONSABILIDAD		5	Paquete seguro
PKG_SGA_SERVICIO		6	Paquete seguro
PKG_SGA_TIPO_DOCUMENTO		1	Paquete seguro
PKG_SGA_TIPO_ESTUDIO_UOP		4	Paquete seguro
PKG_SGA_TITULO		8	Paquete seguro
PKG_SGA_UEN_ADMINISTRADOR		6	Paquete seguro
PKG_SGA_UEN_COMPETENCIA		4	Paquete seguro
PKG_SGA_UEN_COMPONENTE		4	Paquete seguro
PKG_SGA_UEN_CONFIGURACION		9	Paquete seguro
PKG_SGA_UEN_VERSIONCONFIG		8	Paquete seguro
PKG_SGA_UNIDAD_EDUCATIVA		4	Paquete seguro
PKG_SGA_UNIDAD_ENSENANZA		9	Paquete seguro
PKG_SGA_UNIDAD_OPERATIVA		7	Paquete seguro
PKG_SGA_UTIL	PRC_OBT_VERIFICAR_CODIGO	5	Este paquete contiene un procedimiento dinamico que recibe cuatro parametros varchar, y un parametro es recibido desde la UI y el resto de los parametros son enviados a traves de constantes
PKG_SGA_VALIDADOR_TIPO_REQ		1	Paquete seguro
PKG_SOL_SOLICITUD		7	Paquete seguro